Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko

Matjaž Colnarič
Domen Verber

Mikroprocesorji

Učbenik

Laboratorij za sisteme v realnem času

Laboratorij za sisteme v realnem času
Fakulteta za elektrotehniko, računalništvo in informatiko Maribor
UNIVERZA V MARIBORU

Mikroprocesorji

Prva izdaja, pomladi 2000

Avtorja:
Izredni profesor dr. Matjaž Colnarič, dipl.inž.
Asistent dr. Domen Verber, dipl. inž.

Uredil in oblikoval:
Izredni profesor dr. Matjaž Colnarič, dipl.inž. in

Tisk:
Založniško–tiskarska dejavnost
Tehniških fakultet v Mariboru.

Naklada:
?? izvodov

Na podlagi mnenja Republiškega sekretariata za kulturo
Ljubljana je učbenik oproščen prometnega davka.

Predgovor

Kazalo

Poglavje 1

Uvod

Računalniki, ki imajo v zadnjem času skoraj usoden vpliv na naše življenje, izvirajo iz časov tip pred in po drugi svetovni vojni. Prvi uporabni računalniki (npr. Zusejev Z3 ali ENIAC) so nastali ob koncu druge polovice 20 stoletja. Sprva so bili dostopni le vojski in večjim raziskovalnim ustanovam, zaradi napredovanja polprevodniške tehnologije pa so v petdesetih in šestdesetih letih postali manj zahtevni, manjši, zmogljivejši, številčnejši in cenejši. Ta dostopnost je botrovala tudi idejam o uporabi ne samo pri obdelavi številčnih podatkov, temveč tudi za upravljanje zahtevneših procesov. Omejitev pri njihovi uporabi pa je bila v tem, da so bili še vedno razmeroma dragi in veliki ter zato primerni le za omejen spekter aplikacij.

Prva ideja o programirljivem računskem stroju izhaja iz devetnajstega stoletja, ko je Charles Babbage razvil idejo o stroju, ki bi tiskal tablice poljubnih funkcij. Prvi delujoči računalnik, ki ga je zasnoval Konrad Zuse v Nemčiji, datira iz druge polovice tridesetih let; njegove značilnosti so bile binarni številski sistem, plavajoča decimalna vejica in prvi programski jezik.
Po vojni so v ZDA razvili elektronski računalnik ENIAC. To je bil nerodni kolos: tehtal je 30 ton, trošil je 150 kW moči, imel je 18.000 (deloma vodno hlajenih) elektronk.
Računalniki so se lahko bistveno zmanjšali šele, ko so leta 1948 v Bellovih laboratorijih iznašli tranzistor. Sodobna doba v tehnologiji računalnikov se je začela, ko so 1950 pri Texas Instruments in Fairchild Semiconductor izdelali prvo integrirano vezje. Okoli leta 1970 je bilo že mogoče na čipu izdelati nekaj tisoč tranzistorjev.

S pojavom mikroprocesorjev so se razmere pri gradnji naprav, ki jih je potrebno

računalniško krmiliti, tako bistveno spremenile, da so jih okarakterizirali kot "najbolj revolucionarni izdelek v zgodovini človeštva[1]".

Danes si življenja brez mikroprocesorjev v resnici ne moremo več predstavljati. Ne najdemo jih le v osebnih računalnikih; ne da bi se tega posebej zavedali, se z njimi srečujemo malone povsod. Najdemo jih v gospodinjstvu, npr. v mikrovalovni pečici, pralnem stroju, regulaciji centralnega ogrevanja prostorov, telefonih, zabavni elektroniki, fotoaparatu in še marsikje. Kar nekaj jih je že v najpreprostejšem avtomobilu, kjer nastajajo cele lokalne mreže mikroprocesorskih sistemov.

Zelo pomembna je njihova raba v komercialnih in industrijskih sistemih. Tukaj so mikroprocesorji postali nepogrešljiv del kompleksnejših orodij. Na njihovi osnovi so zgrajeni sistemi za vodenje proizvodnje, regulacijski sistemi, roboti, merilne naprave in podobno. Ti sistemi se povezujejo v lokalne mreže, ki so osnova za računalniško povezano proizvodnjo *(CIM)*, v kateri manjše računalnike do krmilnih sistemov v končnih proizvodnih celicah hierarhično nadziramo z močnejšimi. Ti upravljajo proizvodnjo z uporabo podatkovnih baz, ekspertnih sistemov, upoštevajo poslovne in politične odločitve itd.

V krmilnih aplikacijah predstavljajo mikroprocesorski sistemi nadomestek za nekoč diskretne tehniške sisteme, katerih naloga je preslikati vhodne signale v izhodne. Prednosti mikroračunalniškega krmilja pred diskretno logiko so:

- sestavljeni so iz manj komponent, zato je njihov razvoj preprostejši in hitrejši, možnost napake v razvoju manjša, testiranje lažje, cena nižja,
- več različnih naprav lahko uporablja isto elektroniko; standardna mikroračunalniška vezja so univerzalno uporabna,
- zaradi univerzalnih vezij so se sposobni hitro prilagajati spremembam (potrebna je le nova konfiguracija in program),
- zaradi manj elementov, univerzalnih modulov, možnosti samotestiranja (avtodiagnostike) in standardiziranih postopkov je vzdrževanje preprostejše.

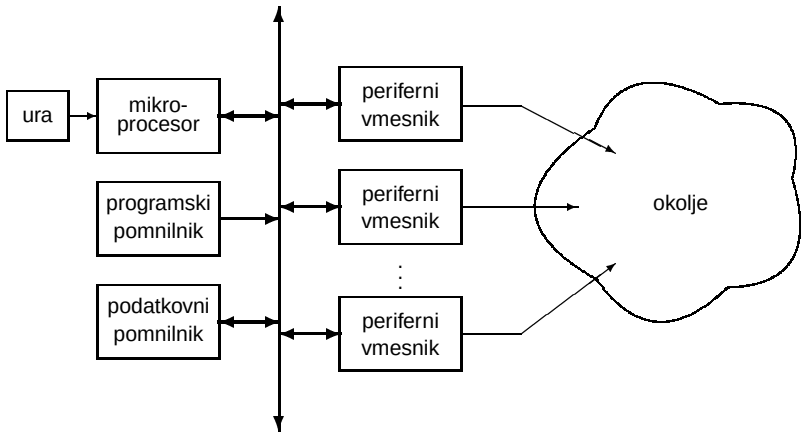
[1] Gordon Moore, ustanovitelj Intela

1.1 Kaj je mikroprocesor?

Mikroprocesor je centralna procesna enota, izdelana v VLSI tehnologiji na enem ali na nekaj celoto sestavljajočih polprevodniških čipih. Vsebuje vsaj krmilno enoto, enoto za obdelavo podatkov *(ALU)* ter najnujnejše notranje shrambe zanje. Poleg tega ima vmesnik za priključitev na sistemsko vodilo, ki predstavlja hrbtenico mikroračunalnika.

Mikroračunalnik ali mikroprocesorski sistem je računalnik na osnovi mikroprocesorja. Sestavljajo ga

- mikroprocesor, glavna enota, ki upravlja delovanje mikroračunalnika. Deluje po taktu, ki ga daje ura;
- programski pomnilnik, od koder mikroprocesor zaporedoma čita strojne ukaze iz programa, jih interpretira in izvaja,
- podatkovni pomnilnik, kjer so shranjeni podatki, ki jih procesor obdeluje, ter
- periferni vmesniki, preko katerih je vzpostavljen stik med mikroračunalnikom in okoljem, v katerem deluje.



Te enote so med seboj povezane z vodilom, preko katerega se pretakajo naslovi, podatki in nekateri krmilni signali.

V mikroračunalniku najdemo tri vrste komponent: strojna (hardware), programska (software) in strojnoprogramska oprema (firmware).

S stališča **strojne opreme** je mikroračunalnik sestavljen iz mikroprocesorja, pomnilnika, vmesnikov za periferne enote in pomožnih sistemskih vezij (generator urinega signala, ura realnega časa, zaščitna vezja, napajanje..). Te enote so med seboj povezane z mikroračunalniškim sistemskim vodilom.

Pomnilnike, ki jih nahajamo v mikroračunalniku, delimo na delovni ali hitri pomnilnik, ki je praviloma izveden v polprevodniški tehnologiji, ter na periferne ali masovne pomnilnike, običajno na magnetnih medijih. V delovnem pomnilniku so naloženi programi, ki jih izvaja mikroprocesor, in podatki, ki jih ti programi obdelujejo.

Med **programsko opremo** spadajo vsi programi, ki jih vgradimo ali naložimo v mikroračunalnik. Delimo jo na *sistemsko in uporabniško*. V sistemsko programsko opremo mikroračunalnikov spadajo operacijski sistem z jedrom in uporabniškim vmesnikom, pomožni programi za razvoj aplikacij, kot so prevajalniki, povezovalniki, razhroščevalniki ipd. K uporabniški ali aplikacijski prištevamo programe, ki jih razvije aplikacijski programer in ki zagotavljajo, da bo mikroračunalnik opravil zahtevane naloge.

Včasih se je težko odločiti, ali neka komponenta spada v strojno ali programsko opremo. To so programi, ki so tovarniško vgrajeni v trajni pomnilnik v mikroprocesorju in s katerimi je realizirano izvajanje strojnih ukazov; imenujemo jo **strojnoprogramska oprema (firmware)**. Ker je ni mogoče spreminjati, jo lahko smatramo kot dano strojno opremo, čeprav je v resnici izvedena kot program.

1.2 Zgodovinski pregled

Rojstvo mikroprocesorja je tesno povezano z družbo Intel. Leta 1968 sta Bob Noyce in Gordon Moore zapustila Fairchild Semiconductors in ustanovila Intel z namenom proizvajati predvsem univerzalne pomnilniške čipe za velike računalnike. Da bi izkoristili svoje razvojne potenciale pa so ponujali tudi snovanje namenskih čipov.

Tako so v začetku leta 1969 dobili naročilo japonske tovarne namiznih računskih

strojev Busicom za sodelovanje pri izdelavi kalkulatorskega čipa. Z Intelove strani je bil za vodenje projekta zadolžen Marcian “Ted” E. Hoff, s strani Busicoma pa Masatoshi Shima.


Marcian Hoff


Federico Faggin


Masatoshi Shima


Stan Mazor

Marcian Hoff je prišel k Intelu kot dvanajsti uslužbenec z Univerze v Stanfordu, kjer se je ukvarjal z gradnjo perifernih vmesnikov za IBM računalnike in je dobro poznal vlogo računalnikov v vodenju procesov. To ga je tudi vzpodbudilo, da se je širše posvetil projektu.

Čeprav je Busicom predlagal izdelavo namenskega čipa za izvajanje kalkulatorskih funkcij, je uspelo Hoffu s podporo vodstva Intela uveljaviti svojo idejo o izdelavi bolj univerzalnega sistema, ki bi bil uporaben za širšo paleto aplikacij. Razvojnemu timu se je pridružil Stan Mazor, ki je v sodelovanju s Shimo zasnoval večji del krmilne enote. Do konca leta 1969 je bil razvoj končan in potrjen s strani Busicoma.

Tedaj je prišlo do težave, da je bilo le malo ljudi sposobnih tudi fizično izdelati tak čip. To delo je sprejel en od najbolj kompetentnih ljudi tega časa, Federico Faggin, ki je pred tem pri Fairchild Semiconductors razvil MOS tehnologijo.

Do konca leta 1970 so izdelali prve prototipe čipov, ki so skupaj tvorili prvi štiribitni mikroračunalnik. Po odpravi “otroških bolezni” so ga leta 1971 vgradili v Busicomov kalkulator, ki je tako postal prva mikroračunalniška aplikacija. Že v naslednjem letu so ga razširili v osembitni 8008, ki je bil vgrajen v prvi programirljivi znanstevni kalkulator firme Seiko. Iz leta 1973 pa datira prav tako na osnovi 8008 zgrajen predhodnik osebnega računalnika, francoski Micral.

Iz 8008 sta Shima in Faggin ob sodelovanju Hoffa in Mazorja razvila 8080, ki je bil veliko zmogljivejši in komercialno prvi uspešni mikroprocesor v enem čipu.

Prvi mikroračunalnik je sestavljal nabor štirih čipov:

- 4001 - programski ROM pomnilnik (256 bajtov in 4 V/I linije),
- 4002 - podatkovni registri (20x4 bitov RAM in 4 V/I linije),
- 4003 - vhodno-izhodne razširitve (10-bitni serijsko-paralelni pretvornik za krmiljenje tipkovnice, tiskalnika, itd.) ter
- 4004 - centralna procesna enota.

Frekvenca ure je bila 750 kHz, ukazni cikel je trajal 8 - 16 urinih. Procesna enota 4004 je vsebovala približno okoli 2300 tranzistorjev. Imela je 4-bitno podatkovno in 12-bitno naslovno vodilo ter 45 ukazov. Čipi so bili pakirani v 14-pinska DIP ohišja. Zmogljivost tega sistema je bila približno primerljiva z zmogljivostjo ENIACa.

Sistem je lahko vseboval en 4004 procesor, do 4Kbajtov programskega pomnilnika (16 čipov 4001), do 1280 štiribitnih registrov podatkovnega pomnilnika (16 čipov 4002), 32 direktno naslovljivih 4-bitnih V/I vrat in neomejeno število izhodnih čipov 4003.
Minimalna konfiguracija je bil 4004 CPU in en 4001 programski pomnilnik s 4 V/I linijami. Za Busicomov kalkulator so potrebovali en 4004, dva 4003, tri 4002 in štiri 4001 module (z dodatkom še enega 4001 pri modelu, ki je omogočal kvadriranje). V sedanji tehnologiji bi za izvedbo vseh teh vezij potrebovali čip s površino precej manj kot 0.1 mm^2.

Pri snovanju vezij se je Faggin odpovedal uporabi najetega računalnika, ker je bilo to predrago; čip je razvil samo s pomočjo logaritemskega računala!
Po istih zahtevah razvito diskretno digitalno vezje z istimi funkcijami je bilo sestavljeno iz 12 čipov s povprečno po 36 do 40 priključki. Njegova prednost pa je bila, da je bilo veliko hitrejše.

Od tedaj pri Intelu v povprečju vsake tri leta nastane nova generacija mikroprocesorjev.

V tem času se na tržišču pojavijo tudi drugi proizvajalci mikroprocesorjev; to so bili Texas Instruments TMS-1000 (1973), Rockwell s 4– (1972) in 8-bitnim mikroprocesorjem (1974), National IMP 8 (1973), RCA Cosmac (1974). 1975 je Faggin, tokrat za svojo novo družbo Zilog, razvil najbrž najboljčisti mikroprocesor tiste dobe, Z80.

V 1973 se pojavi tudi Motorola z enim najuspešnejših mikroprocesorjem MC6800 s 5000 tranzistorji in frekvenco ure 1 (kasneje 1,5 in 2) MHz. Ta mikroprocesor je imel popolno družino perifernih vmesnikov, od paralelnih, serijskih, števcev, do namenskih krmilnikov. Družino so kasneje nadgrajevali z novimi, zmoglji-

vejšimi mikroprocesorji, ki so se razen po novi tehnološki izvedbi odlikovali tudi po vedno več vgrajenih funkcijah, kot so generator sistemske ure, programski pomnilnik, RAM, nove funkcije za obdelavo podatkov. Tako je nastal prvi Motorolin mikrokrmilnik 6801, kateremu je sledil 6805, arhitektura, ki je na tržišču še danes v vedno novih tehnoloških oblikah.

Zgodba se nadaljuje s 16-bitnimi mikroprocesorji. Spet je prvi Intel z 8086 leta 1978, ki je imel 29.000 tranzistorjev. Motorolin 68000, ki mu je sledil leto kasneje, je imel po naključju ravno 68.000 tranzistorjev in je bil precej popolnejši od 8086. Druga zgodnja 16-bitna mikroprocesorja sta bila PACE (National, 1973) in Z8000 (Zilog, 1979).

V osemdesetih letih se je nadaljeval hiter razvoj 16-bitnih mikroprocesorjev. Motorola predstavi 68008 z osembitnim zunanjim vodilom, s čemer se je zmanjšala kompleksnost zunanjih vezij ob istem elegantnem programskem modelu, in 68010 kot izboljšan 16-bitni mikroprocesor. Intel razvije 80186 in 80286.

Leta 1984 sta nastala prva prava 32-bitna mikroprocesorja, 80386 in 68020, vsak s svojim aritmetičnim ko-procesorjem, 80387 in 68881. Zorenje se je nadaljevalo: 1987 izide 68030 z 270.000 tranzistorji na 20 MHz, leta 1989 pa 80486 in 68040, ki sta imela podobne podatke: čez milijon tranzistorjev in frekvenco ure 25 MHz. Ta razvoj so spremljale novosti na področju mikrokrmilnikov (8051, 6811), ki so postali zelo razširjeni za uporabo v vgrajenih krmilnih sistemih.

V tem času se začne razvijati tudi alternativni pristop k razvoju mikroprocesorjev, RISC. Leta 1986 se pojavi MIPS R2000, v širšem časovnem obdobju mu sledijo Sun Sparc, AMD-jev 29000, Acorn-ov ARM, Digitalova Alpha in HPjev PA-RISC. RISC arhitektura prodre tudi v vgrajene sisteme (Microchip - PIC, Atmel,..)

Devetdeseta leta so zaznamovana predvsem z dvema mikroprocesorjema; to sta Intelov Pentium in Motorolin PowerPC, ki se pojavita 1993. Leta 1995 je Motorola prekinila linijo 68000, ki je bila na tržišču celih šestnajst let, in zapolnila vrzel z mikroprocesorjem ColdFire, ki je zgrajen na RISC principih in 68000 programskem modelu.

Današnji trendi razvoja se vse bolj približujejo 64 bitni arhitekturi. Najbolj značilna predstavnika te nove generacije sta Intelova arhitektura IA-64 in Sunov UltraSPARC–III. Pripravlja pa se tudi 64 bitna različica procesorja PowerPC.

1.3 Vrste mikroprocesorjev

Mikroprocesorje prvenstveno delimo po **dolžini besede** na (1,4),8,16,32,64 *bitne*. Pri tem je dolžina besede običajno slabo definiran pojem. Razlikovati moramo med dolžino *notranje* in *zunanje* besede. Dolžina notranje besede je določena z arhitekturo mikroprocesorjev (širino notranjih poti, velikostjo registrov, dolžino besede, ki jo lahko v enem koraku obdela aritmetična enota ipd.), dolžina zunanje besede pa s številom podatkovnih priključkov na čipu. Notranja beseda je običajno daljša. Pri krajši notranji besedi je treba operacije za prenos podatkov večkrat izvajati, kar pomeni izgubo časa. Po drugi strani pa široka podatkovna vodila poleg večje velikosti čipa zahtevajo tudi več in zahtevnejša dodatna vezja (vmesnike, dekodirnike itd.); primera kompromisov sta mikroprocesorja 8088 in 68008, ki imata 32 bitno notranje in 8-bitno zunanje vodilo.

Za označevanje procesorjev jemljemo običajno zunanje poti, še popolnejše pa je omeniti oboje. Važen podatek je tudi dolžina naslova, ki omejuje količino neposredno naslovljivega pomnilnika.

Po **kompleksnosti nabora ukazov**, načinov naslavljanja operandov in po sorodnih značilnostih delimo mikroprocesorje na *CISC (Complex Instruction Set Computers)* in *RISC (Reduced Instruction Set Computers)*.

Nastanek RISC mikroprocesorjev je bil odgovor na hitro naraščajočo kompleksnost zaradi povečevanja števila ukazov in načinov naslavljanja ob pojavu 16-bitnih mikroprocesorjev. Čeprav je bilo programiranje v zbirnem jeziku zaradi tega vedno elegantnejše, je postajalo dekodiranje in izvajanje ukazov vedno počasnejše. Na univerzah Berkeley in Stanford so raziskovali, kaj bi pomenilo, če bi mikroprocesor izvajal le tiste ukaze, ki so najpogostejši in neizogibno potrebni za izvedbo osnovnih funkcij. Na osnovi statističnih rezultatov o pogostosti posameznih ukazov so sestavili njihov najožji možni nabor, s čemer se bistveno zmanjša kompleksnost in posledično poveča hitrost pripadajočega dekodirnika ukazov. Simulacije delovanja takšnega mikroprocesorja so pokazale, da se program, napisan v višjem programskem jeziku, ustrezno preveden precej hitreje izvaja na RISC kot na CISC procesorju.

V zadnjem času se pri zahtevnejših mikroprocesorjih meja med RISC in CISC zabrisuje. Tudi CISC mikroprocesorji (npr. PENTIUMi) vsebujejo že veliko naprednih arhitekturnih rešitev, ki izvirajo iz RISC arhitektur, npr. cevovodne arhi-

tekture, napovedovanje pogojnih skokov, ipd.)

Statistične raziskave pogostosti različnih operacij v računalniških programih so že razmeroma zgodaj izvajali D. Knuth (1971, na FORTRANskih programih), Wortman (1972, v PL/1), A. Tannenbaum (1978, v Pascalu) in D. Patterson (1982 v C in Pascal).
Rezultati teh raziskav so pokazali, da 85% programskih konstruktov tvorijo preprosti izrazi, klici procedur in pogojni stavki. Pri izrazih prevladujejo (80%) izrazi tipa X:=vrednost. Le 15% izrazov vsebuje na desni strani operator, torej le 5% dva ali več operatorjev.
Za takšno zahtevnost ukazov ni potrebno podpirati širokega nabora ukazov ali načina naslavljanj. Na osnovi teh rezultatov so že leta 1975 pri IBM zasnovali prvi RISC procesor, čeprav ga niso objavili do leta 1982. Leta 1980 sta začela z raziskavami in razvojem RISC mikroprocesorjev David A. Patterson in Carlo Séquin na University of California v Berkeleyu in razvila čipa RISC I in RISC II, ki sta predstavljala osnovo za danes najbrž najbolj razširjene RISC mikroprocesorje, Sun-ove SPARC. Nekaj kasneje je John Hennessy na Stanfordu razvil mikroprocesor MIPS.

Posebni avtonomni tipi mikroprocesorjev: z razvojem tehnologije je bilo mogoče v mikroprocesor vgraditi tudi nekatera druga vezja, ki jih je bilo prej potrebno dodati zunaj. To bili sprva npr. generatorji ure, razni vmesniki za vodilo, dekodirniki naslovov za izbiro raznih vezij ipd. Kasneje so začeli dodajati še pomnilnik, najprej nekaj deset ali sto celic RAM-a, nato ROM, ki so ga programirali z masko, in še kasneje tudi EPROM, za razvoj ter manjše serije. Končno je tehnologija omogočila, da so v čip vgradili tudi cela periferna vezja, paralelne in serijske vmesnike, analogne pretvornike ipd. Mikroprocesorji s takšnimi dodatki se imenujejo *mikrokrmilniki* ali enočipni računalniki (single–chip computers).

Podobno so zgrajeni tudi *signalni procesorji*, ki poleg aparaturnih dodatkov vsebujejo tudi posebne vgrajene ukaze za neposredno digitalno obdelavo analognih signalov. Posebej zmogljivi so signalni procesorji, ki so namenjeni za zabavno elektroniko. Ti obdelujejo komprimirane video in zvočne signale v realnem času.

1.4 Pregled tehnologije

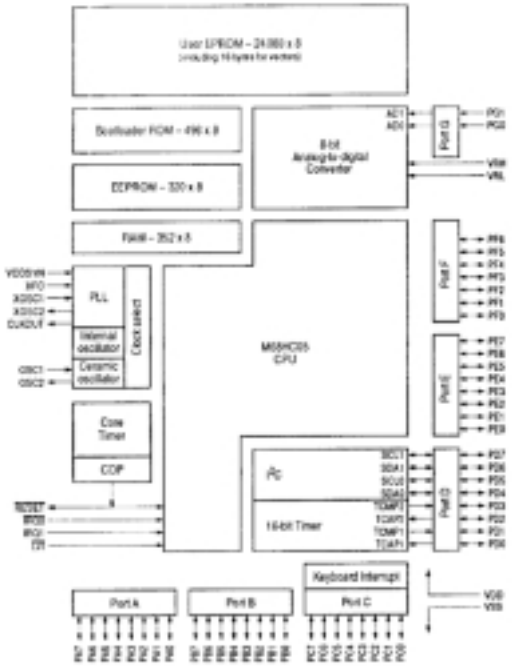
Nastanek mikroprocesorja je bil pogojen z dovolj visoko stopnjo razvoja polprevodniške tehnologije, ki je morala omogočati zelo visoko stopnjo integracije (VLSI); že prvi mikroprocesor je imel 2300 tranzistorjev na površini 12 mm^2.

Kot substrat se običajno uporablja silicijev kristal. Že dalj časa lovi korak z njim tudi galijev arzenid GaAs; njegove prednosti so manjše dimenzije kristalne mreže

Ena od najbolj razširjenih družin mikrokrmilnikov je Motorolina 68HC05. Osnova njene arhitekture je podobna njihovemu prvemu osembitnemu mikroprocesorju 6800. Seveda se je od takrat tehnologija bistveno spremenila in sedaj omogoča zelo zmogljiv nabor funkcij na enem samem čipu.

Različne verzije mikrokrmilnika 68HC05 vsebujejo različne tipe pomnilnika (ROM, EPROM, EEPROM, RAM) veliko število univerzalnih in namenskih digitalnih vhodov in izhodov, serijske komunikacijske porte, števce in analogno/digitalne pretvornike.

Vgraditev mikrokrmilnikov v vodene naprave je praviloma zelo preprosta: ko na razvojni opremi, ki običajno teče na osebnem računalniku in jo pogosto dobimo kar zastonj, izdelamo in simuliramo program, ga vpišemo v mikrokrmilnikov programski pomnilnik. Ko priključimo še napajanje ter vhodne in izhodne signale, je naprava pripravljena za testiranje. Za profesionalno rabo obstajajo seveda zahtevnejša orodja, kot so analizatorji, emulatorji in podobno.



in s tem povezane izredno velike hitrosti signalov ter neobčutljivost na razna sevanja. Slednje je predstavljalo resen problem pri uporabi silicijevih polprevodnikov v vojaški tehnologiji. Poleg tega je GaAs neprimerno manj občutljiv na temperaturo ($-200^\circ C \ldots +200^\circ C$) in ima dobre fotoelektrične lastnosti, kar omogoča neposredno implementacijo optičnih sprejemnikov in oddajnikov na čipu.

Poleg teh dveh tehnologij se v zadnjem času pojavljajo tudi kombinacije silicija z germanijem in silicija na izolatorju.

Mikroprocesorji so najpogosteje izdelani v bipolarni ali MOS (Metal Oxide Semicondictor) tehonologiji. Razvoj je tekel od bipolarne, preko n-MOS, p-MOS

do CMOS inačic. Sedanja CMOS tehnologija omogoča zelo visoke stopnje integracije, večjo zanesljivost in majhno porabo moči, kar je pomembno zaradi termičnih izgub (lažje hlajenje). Je dražja, a hitrejša od navadne MOS tehnologije, a počasnejša od bipolarne. Za najzmogljivejše čipe zato uporabljajo bi-CMOS tehnologijo: večina čipa je izvedena v CMOS, časovno kritični deli pa v bipolarni tehnologiji.

Ključni cilj pri razvoju polprevodniške tehnologije je miniaturizacija. Manjših elementov je možno realizirati več na komercialno upravičljivi površini čipa, povezave med njimi so krajše in zato signali hitrejši, napetosti in s tem poraba energije ter posledično gretje čipa manjše.

Najmanjša dimenzija elementa, ki ga lahko izvedemo v čipu, je odvisna od kvalitete fotolitografije: optičnih postopkov, s katerimi na fotosenzibilne premaze na čipu z laserjem projeciramo masko; v nadaljnjem procesu s kemijskimi postopki na teh mestih nastanejo različne komponente vezij. Trenutna tehnologija se približuje velikosti 0.1μ. S tem bodo lahko dosegli do milijarde tranzistorjev načipu običajne velikosti.

Površina čipa, ki se trenutno uporablja, je okoli 1 cm^2. pri večjih površinah je statistično večja verjetnost nečistoč v kristalni strukturi in s tem manjši izplen pri proizvodnji.

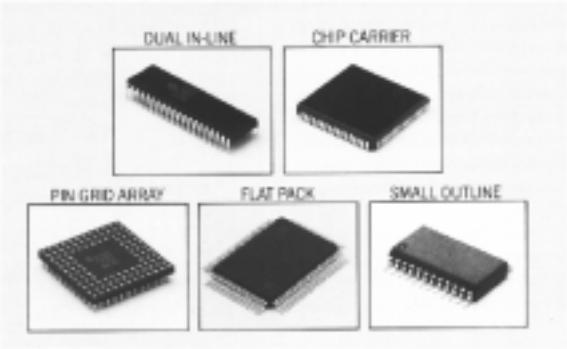
Tudi hitrost delovanja se je dvignila tako visoko, da omogoča urne frekvence do 1GHz. Pri tem pride do nepričakovanih problemov: zaradi velikih hitrosti in čim tanjših povezav (zaradi varčevanja s prostorom) hitrost potovanja signala po aluminijevih povezavah ne zagotavlja več, da bo le-ta v enem urinem ciklu pripotoval od enega do drugega vogala čipa (okoli 1cm). V perspektivi se bo problem le še zaostroval: delovanje vrat bo hitrejše, hitrost signalov pa manjša. Zato nadgrajujejo aluminijeve povezave z naparitvijo bakrene prevleke.

1.5 Trendi razvoja

V zadnjem času polprevodniška tehnologija ni več ozko grlo za napredek mikroprocesorjev. Pri takšni neizmerni količini tranzistrojev, ki jih znamo izdelati na čipu, se pojavljajo problemi, kaj sploh narediti z njimi; kako zasnovati tako vezje, ki jih bo lahko izkoristilo in predvsem kako ga stestirati. Najkompleksnejši mi-

Mikroprocesorji so pakirani v različna ohišja glede na njihov namen, velikost, kompleksnost in zahtevane termične in mehanske lastnosti. Starejši so imeli obliko DIL (dual-in-line) s 40 do 64 priključki (pini) z razmikom med njimi 2,54 mm. Zaradi povečanega števila priključkov so sedaj ohišja največkrat kvadratna (Quad-Pack) različnih izvedb:
– SMT (Surface-Mount-Technology, tudi Flat-Pack), ki se prispajkajo na tiskano vezje brez lukenj,
– LCC (Leadless Chip Carrier, bočni kontakti; vtaknejo se v podnožje s priključki na robovih),
– PGA (Pin-Grid-Array - z matriko priključkov na spodnji strani, ki se vtakne v podnožje; s tem dosežemo največjo gostoto priključkov)
ali drugačnih, manj standardnih izvedb.

Ohišja so izdelana iz plastike, keramike in kovine. Pogosto je potrebno zagotoviti dobro odvajanje toplote. V tem primeru se na ohišje namestijo hladilna rebra ali ventilatorji. Obstajajo procesorji, ki jih je potrebno hladiti s tekočim helijem!



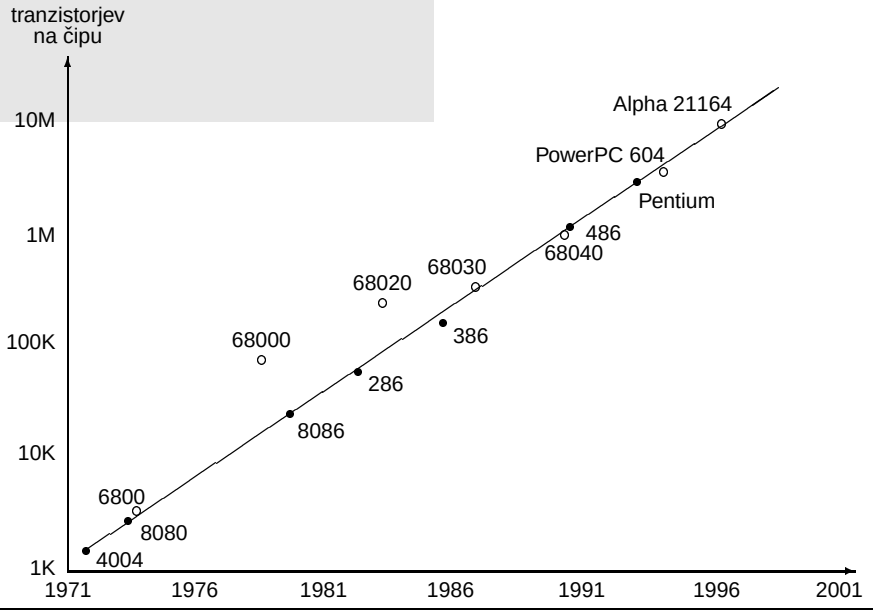
kroprocesorji danes zavzemajo okoli 10 milijonov tranzistorjev, medtem ko bo kmalu mogoče narediti čip s stokrat večjo gostoto.

V prihodnosti lahko pričakujemo nove inventivne arhitekture mikroprocesorjev. Ena od možnosti je povezava mikroprocesorjev s pomnilniki; na to nas napeljuje več dejstev:

- medtem ko kompleksnosti mikroprocesorjev nad določeno mejo še ne moremo obvladovati, je to pri pomnilnikih trivialen problem. Ti so sestavljeni iz velikega števila enakih majhnih celic, katerih število lahko poljubno večamo brez povečanja kompleksnosti zasnove.
- za omilitev ozkega grla, ki nastopa pri prenosu podatkov po vodilu med procesorjem in pomnilnikom, se poslužujemo hitrih predpomnilnikov (cache). Zaradi znanih problemov se pogosto dogaja, da podatkov v njih ne najdemo (cache miss). Zato je smiselno poiskati drugačne rešitve problema ozkega grla.

Moorov zakon

Leta 1970 je dr. Gordon Moore, soustanovitelj in predsednik družbe Intel med pripravljanjem predstavitvenega govora opazil, da točke, ki jih je vrisal v diagrame razmerij med časom nastanka ter lastnostmi mikroprocesorjev (kapaciteto, velikostjo, hitrostjo in ceno) v logaritemskem merilu, tvorijo premico. Na osnovi teh grafov je ugotovil, da se zmogljivosti mikroprocesorjev podvojijo vsako leto in pol do dve leti. Tako je tudi napovedal razvoj mikroprocesorjev do danes; čeprav mu je tedaj malokdo verjel, so se njegove napovedi presenetljivo uresničile.
Kot primer podajamo na spodnjem grafu graf kapacitet mikroprocesorjev v odvisnosti od leta njihovega nastanka.



Da bi se izognili prenosu podatkov po vodilu, lahko procesor implementiramo kar v samem pomnilniku; število tranzistorjev to že omogoča. Komponente procesorja lahko sedaj neposredno posegajo v pomnilnik po podatke in ne preko vodila. V tem primeru je cache nepotreben. Takšno kombinacijo pomnilnika in procesorja so poimenovali **iRAM** (Intelligent RAM).

Druga ideja, kako izrabiti veliko število tranzistorjev, je uresničena v prototipnih izvedbah tako imenovanega **"surovega stroja"** (Raw Machine). To je neke vrste

nadgradnja FPGA vezij, ki so že dolgo znana v snovanju digitalnih vezij. Sestavlja ga množica procesorjev zmerne zmogljivosti, ki so med seboj povezani z dinamično rekonfigurabilnimi povezavami.

Ideja je ta, da prevajalnik glede na program zasnuje tako arhitekturo multiprocesorskega sistema, ki bo optimalno rešila problem. Na posamezne procesorje preslika določen del programa in obenem ustrezno konfigurira povezave med njimi.

Direktor National Semiconductors je strokovno javnost presenetil z izjavo, da je doba računalništva končana. Kot paralelo je navedel primer iz elektromehanike: nekoč so bili elektromotorji zahtevni, veliki, težki, dragi in so predstavljali glavni del naprave. Danes so njeni samoumevni deli in jih nihče posebej ne opazi. Podobno se bo dogajalo z računalniki: njihova prisotnost bo vse bolj prikrita v funkcionalnosti izdelka, v katerega so vgrajeni.

Doba, ki prihaja, bo torej doba uporabe računalnikov. Ti se bodo vse bolj prilagajali človekovemu vsakdanjemu življenju in jih praktično ne bo več opaziti. Tako bodo npr. osebni računalniki združeni s telefonom (povezava v omrežje), televizorjem (prikazovalnik), raznimi napravami, ki jih bodo nadzirali (npr. ogrevanje), in bodo tako predstavljali upravni in komunikacijski center doma. Obstajajo tudi prototipi računalnikov, vgrajeni v oblačila, ki človeku pomagajo pri vsakdanjih dejavnostih, kot je npr. komunikacija navigacija, itd.

Tehnike navidezne resničnosti bodo pomagale npr. pri servisiranju raznih naprav, pri čemer bodo nanje projicirale njihove notranje nevidne dele, načrte in drugo dokumentacijo. Komunikacija človek-stroj bo potekala na naravni način z običajnimi komunikacijskimi mediji - govor, projekcija slike v očala ipd.

Poglavje 2

Arhitektura mikroprocesorjev

Za razumevanje delovanja mikroprocesorjev bomo v tem poglavju postavili preprost model, v katerem bomo upoštevali le njihove najbolj splošne in pomembne funkcije. Resnična izvedba tipičnega mikroprocesorja je veliko bolj kompleksna; gradnikov je več, v njih se prepletajo funkcije različnih funkcionalnih enot. V 9. poglavju bo ta model razširjen s podrobnostmi, ki jih najdemo v sodobnih mikroprocesorjih. Tam bodo tudi podrobneje opisane izbrane sodobnejše mikroprocesorske arhitekture.

Osnovne funkcije mikroprocesorja lahko povzamemo zelo na kratko: prenos, hranjenje in obdelava podatkov, kakor zahteva uporabniški program. Gledano funkcionalno, sestavljajo model mikroprocesorja tri enote: krmilna enota, aritmetično-logična enota in nabor registrov.

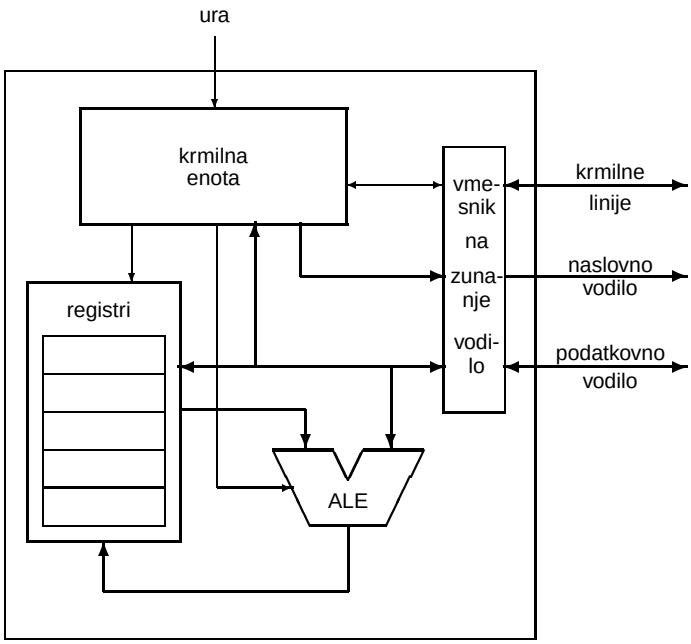
2.1 Model mikroprocesorja

Arhitekture mikroprocesorjev se zelo razlikujejo med seboj, večina pa jih temelji na von Neumannovem modelu, kjer se podatki in programska koda nahajajo skupaj v istem pomnilniškem naslovnem področju. To je bila nesporno ustrezna in zelo koristna rešitev v časih, ko je obdelava ukazov zahtevala bistveno več časa kot njihov prenos. V zadnjem času se je ozko grlo preselilo na slednje. Von Neumannova arhitektura je postala šibka točka, ki jo predvsem v RISC mikroprocesorjih

nadomešča harwardska: ta ima ločeni pomnilniki področji in podatkovne poti za podatke in ukaze. Prenos enih in drugih se zato lahko vrši popolnoma vzporedno.

Slika obeh modelov ? (je podana v 1. poglavju ?)

Arhitekturo mikroprocesorja lahko v grobem razdelimo na tri glavne sestavne dele: krmilno enoto, aritmetično-logično enoto in nabor registrov.



Oznake slik ?

Najpomembnejši del mikroprocesorja predstavlja krmilna enota, ki usklajuje delovanje posameznih delov mikroprocesorja po navodilih programske kode. Krmilna enota čita ukaz za ukazom iz programa, zapisanega v strojni obliki, jih dekodira in preko množice krmilnih signalov krmili njihovo izvajanje.

Za izvajanje operacij nad podatki, ki jih obdeluje program, skrbi aritmetično-logična enota (ALE). Ta je sposobna opraviti različne matematične in logične operacije nad podatki različnih tipov.

Registri služijo za začasno hranjenje in obdelavo podatkov in njihovih naslovov. Izvedeni so kot zelo hitre pomnilne celice (običajno 5 do 10 krat hitrejše od zu-

nanjega – delovnega). Imamo več vrst registrov, od katerih so nekateri dostopni programerju, drugi pa so namenjeni internemu delovanju mikroprocesorja. Nekateri registri nastopajo tudi kot del krmilne oziroma aritmetične in logične enote.

Posamezne enote mikroprocesorja so med seboj povezane preko več internih vodil. Širina teh vodil običajno po širini ustreza velikosti registrov. Prav tako je mikroprocesor je z okolico povezan preko zunanjih vodil, ki jih pri večini mikroprocesorjev sestavljajo:

- *naslovno vodilo*, ta določa naslov pomnilniške lokacije, do katere želi mikroprocesor dostopati;
- *podatkovno vodilo*, preko njega poteka izmenjava vsebine (programskih ukazov in podatkov) med registri in celicami pomnilnika, torej v ali iz mikroprocesorja; ter
- *krmilne linije*, ki krmilijo komunikacijo z okolico (povejo na primer, ali želi mikroprocesor brati ali pisati v pomnilniško področje).

Zaradi tehnoloških omejitev so bile širine zunanjih vodil pri starejših mikroprocesorjih pogosto manjše od širine internih vodil. To je pomenilo, da je bilo potrebno za prenos ene besede izvesti dva ali več bralnih oz. pisalnih ciklov. V sodobnih mikroprocesorskih arhitekturah težimo k čim večji prepustnosti podatkov in nimajo takšnih omejitev. Zunanje in notranje vodilo sta povezana preko posebnega vmesnika, katerega delovanje je pod nadzorom krmilne enote.

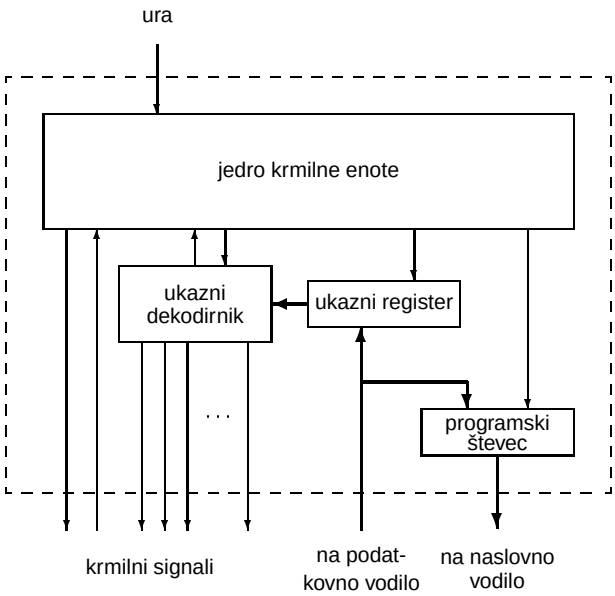
Nekateri preprostejši mikroprocesorji in mikroračunalniki imajo programski in podakovni pomnilnik vrgajen na isti silicijevi rezini. Takšni mikroprocesorji seveda zunanjega naslovnega in podatkovnega vodila ne potrebujejo.

Podrobneje bo o vodilih govora v 4. poglavju.

2.1.1 Krmilna enota

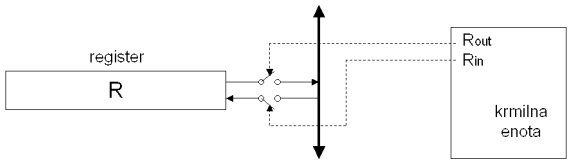
Najpomembnejši del mikroprocesorja je krmilna enota. To je sinhrono vezje, ki deluje po taktu sistemske ure (system clock). Preko posebnih krmilnih linij, po katerih se ne prenašajo podatki, upravlja z delovanjem posameznih pod-enot in krmili komunikacijo z okolico mikroprocesorja.

Na spodnji sliki je prikazan primer, ko krmilna enota krmili dostop do enega izmed registrov. Potrebna sta dva signala. Prvi skrbi za prenos podatkov v register (R_{in}), drugi pa za prenos podatkov iz registra (R_{out}).



Podobni signali vodijo v ALE in določajo katera aritmetično/logična operacija se mora izvesti.

Na naslednji sliki je prikazana podrobnejša zgradba krmilne enote.



Osnovo krmilne enote predstavlja njeno jedro, ki vodi njeno delovanje. Jedro je tesno povezano z dekodirnikom ukazov ter ukaznim registrom. V ukazni register se zapiše koda ukaza, ki ga mora mikroprocesor izvesti, dekodirnik ukazov pa v skladu s tem ukazom generira krmilne signale in tako poskrbi za njegovo izvedbo. Naslov pomnilniške lokacije, v kateri se nahaja naslednja koda ukaza za izvajanje, je podan v programskem števcu (PŠ). Vrednost tega registra spreminja običajno

le krmilna enota. Praviloma deluje tako, da se programski števec med dekodiranjem vsakega ukaza avtomatsko poveča in kaže na naslednji ukaz v programu. Programer spreminja njegovo vrednost samo preko ukazov skokov in klicev podprogramov. Ob vklopu se vrednost PŠ postavi na vnaprej definirano vrednost in kaže na začetek zagonskega dela programa.

2.1.2 Aritmetično-logična enota

Za izvajanje operacij nad operandi ukazov mikroprocesorja skrbi *aritmetična/logična enota, ALE* (Arithmetic/Logic Unit). Sodobni procesorji imajo običajno več aritmetično-logičnih enot, ki skrbijo za računanje z različnimi tipi podatkov: ene skrbijo za celoštevične podatke, druge pa za računanje s podatki s pomično vejico. Nekateree ALE so sposobne izvajati tudi kompleksnejše računske operacije, kot so npr. logaritmi, kotne funkcije ipd. Nekatere od njih so prirejene tudi za izvajanje operacij posebnega tipa npr. za obdelavo signalov,slik ipd.

ALE je običajno izvedena je kot čisto preklopno vezje, ki operira nad enim ali dvema vhodnima podatkoma in generira tretjega – rezultat. Krmilna enota s krmilnimi signali izbira poljubne aritmetične ali logične operacije nad njimi, ALE pa poleg rezultata generira še dodatne informacije v obliki t.i. zastavic (flags) o tem, ali je bil rezultat zadnje operacije negativen ali enak nič, ali je pri operaciji prišlo do prekoračitve območja in podobno. Glede na te informacije lahko krmilna enota izvede ali ne pogojne skoke in podobne ukaze. Podrobnejši opis operacij, ki jih lahko izvaja ALE, in pomenov zastavic je podan v naslednjem poglavju.

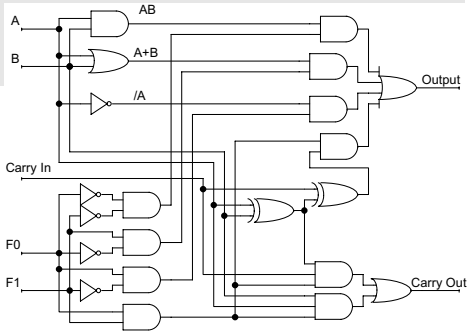
2.1.3 Registri

Registri so začasne shrambe za podatke, največkrat izvedene na samem čipu in z zelo kratkim dostopnim časom. Njihova izbira poteka neposredno s krmilnimi signali in ne z naslavljanjem kot pri zunanjem pomnilniku.

V mikroprocesorju nastopa več vrst registrov; v grobem jih delimo na programerju dostopne in nedostopne. Osnovne skupine registrov, ki jih srečamo v skoraj vseh mikroprocesorjih so naslednje:

- *Podatkovni registri*, ki služijo za vmesno hranjenje podatkov med obde-

Tukaj prikazujemo model logičnega vezja, ki predstavlja preprosto enobitno aritmetično/logično enoto; ta zna izračunati logično konjunkcijo (IN), disjunkcijo (ALI), negacijo in aritmetično vsoto dveh bitov A in B z upoštevanjem prenosov. Želeno funkcijo bi krmilna enota izbirala s krmilnima signaloma F0 in F1. S paralelno vezavo bi lahko dobili ALE za poljubno dolžino besede. Prave ALE so seveda neprimerno bolj kompleksne.



F_0	F_1	Output	CarryOut
0	0	$A \wedge B$	0
0	1	$A \vee B$	0
1	0	$\neg A$	0
1	1	A+B+CarryIn	CarryOut

lavo. Tipični podatkovni registri so bili pri zgodnjih mikroprocesorjih t.i. akumulatorji ali zbirni registri; ti so služili pri aritmetično/logičnih operacijah za nosilce enega vhodnega operanda in rezultata. Namesto akumulatorjev imajo sedaj mikroprocesorji več neodvisnih podatkovnih registrov. Podatkovni registri običajno hranijo celoštevilčne informacije, v novejših miproprocesorjih pa srečamo tudi posebno skupino podatkovnih registrov, ki so namenjeni za izvajanje operacij nad števili s pomično vejico.

- *Naslovni registri* vsebujejo naslove operandov ali dele, iz katerih se le–ti izračunajo. Naslovni registri nastopajo v različnih načinih naslavljanja, ki bodo podrobneje opisana v naslednjem poglavju.

- *Posebni registri* imajo v mikroprocesorjih ozko specialne funkcije. Nekateri med njimi, t.i. specialni registri, so programerju nedostopni oz. le deloma dostopni in služijo za pomožne shrambe. K tem prištevamo zgoraj opisana pomožna registra v procesni enoti, ukazni register in programski števec.

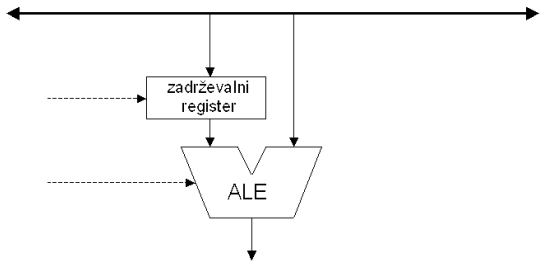
V zgodnješih tipih mikroprocesorjev so določeni registri nastopali le v določenih ukazih mikroprocesorja. Danes težimo k čim večji univerzalnosti. Tako v sodobnih mikroprocesorjih ne ločimo več podatkovnih in naslovnih registrov in lahko vse splošno namenske registre uporabljamo za ene ali druge namene. Razvijalci

CISC mikroprocesorjev so težili k čim večjemu številu registrov, da bi lahko hitro posegali do čim več spremenljivk. Vendar je to tedanja tehnologija to onemogočala.Dodatni problem je predstavljala tudi širina ukaza. V osem bitnih mikroprocesorjih lahko identifikaciji določenega registra namenimo le manjše število bitov. Ko je tehnologija omogočila na čipu realizirati veliko registrov in ko se je velikost operacijske kode povečala, je nastopil problem razporejanja spremenljivk na registre: kako čim bolj optimalno razporediti spremenljivke programa, posebej v višjem programskem jeziku, v posamezne registre, tako da bo dostop do njih kar najhitrejši. Optimizacija uporabe registrov je zapletena, posebej, ko gre za multiprogramiranje ali rekurzivne klice programov. Zaradi tega jo običajno prepustimo prevajalniku.

2.2 Prenos podatkov znotraj mikroprocesorja

Prenos podatkov znotraj mikroprocesorja poteka preko internih vodil. Glede na izvedbo ločimo dva načina prenosa podakov:

- *Skupno notranje vodilo.* Pri takšni organizaciji poteka celotni promet znotraj mikroprocesorja po eni skupini povezav. Težave nastopijo v primeru,če je potrebno izvesti operacijo ki zahteva več kot en operand (npr. seštevanje dveh števil). V tem primeru moramo izvesti operacijo v dveh korakih kot je to prikazano na spodnji sliki.



  V prvem koraku vsebino prvega operanda prenesemo v posebni zadrževalni register s čemer ga pripravimo za izvedbo operacije. V drugem koraku po vodilu prenesemo vrednost drugega operanda in izvedemo operacijo.

- *Ločena notranja vodila* Seveda je realizacija prenosa podatkov po skupnem vodilu neučinkovita, vendar se je pri zgodnejših mikroprocesorjih zaradi

tehnoloških omejitev pogosto uporabljala. Zaradi težnje po čim hitrejšem delovanju, uporabljajo vse novejše mikroprocesorske arhitekture več notranjih vodil za komunikacijo med posameznimi segmenti mikroprocesorja. Zgled uporabe takšnega pristopa bo podan v naslednjem podpoglavju.

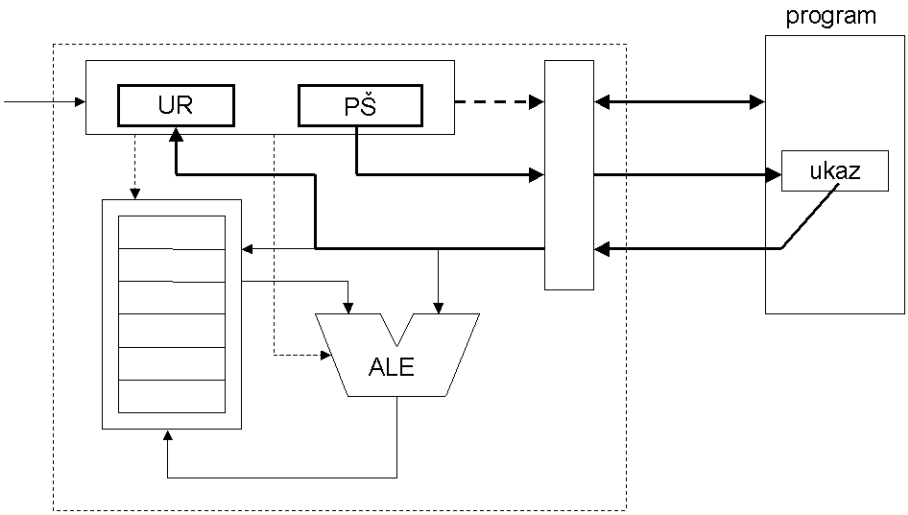
2.3 Delovanje mikroprocesorja

V tem podpoglavju si bomo na primeru hipotetičnega mikroprocesorja ogledali osnovne korake v njegovem delovanju. Model smo zaradi lažjega razumevanja zelo poenostavili. Čeprav sodobni mikroprocesorji temeljijo na podobnem modelu delovanja, pa je njihova zgradba veliko kompleksnejša. Spodaj opisane faze delovanja se v vseh sodobnih mikroprocesorjih medsebojno prepletajo, več faz (nad različnimi ukazi) se izvaja vzporedno, itd. Najprej bo delovanje mikroprocesorja predstavljeno bolj splošno, na koncu pa bomo podrobneje predstavili še podrobnejši primer izvedbe hipotetičnega mikroprocesorja.

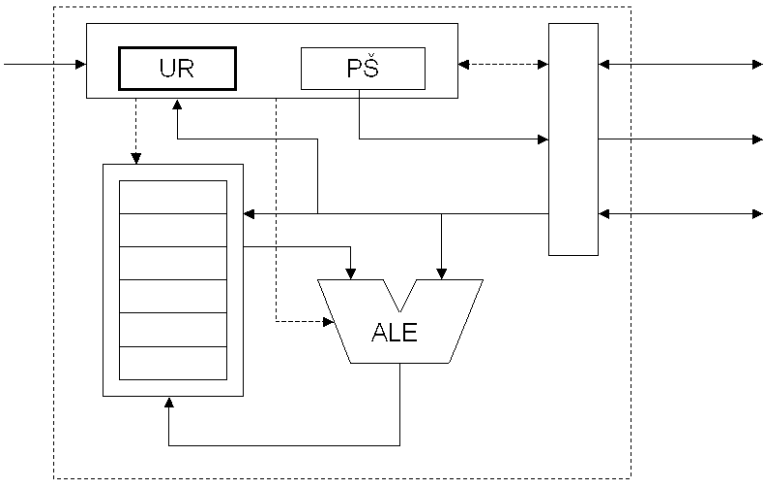
2.3.1 Faze delovanja

Izvajanje ukazov mikroprocesorja lahko razdelimo na več zaporednih faz:

1. *Prevzem ukaza*
   V tej fazi jedro krmilne enote poskrbi, da se vrednost programskega števca prenese na naslovno vodilo in s podatkovnega vodila prečita vrednost z lokacije, na katero kaže. Prečitana vrednost se prenese v ukazni register. Po opravljenem prenosu se vrednost programskega števca poveča tako, da kaže na naslednji ukaz.

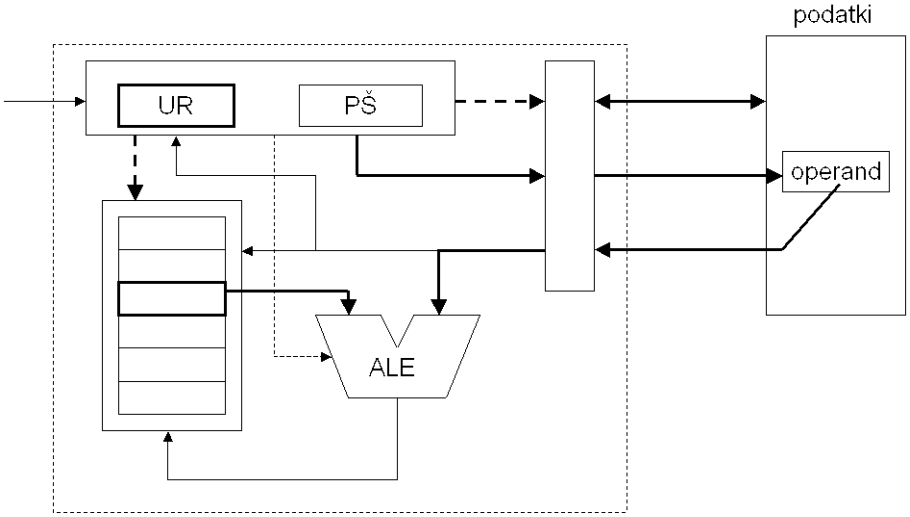


2. *Dekodiranje ukaza*
   Vrednost, zapisano v ukazni register, prevzame ukazni dekodirnik. Ukaz se prevede v sekvenco ustreznih krmilnih signalov, ki vklapljajo in izklapljajo posamezne dele mikroprocesorja. Ta sekvenca signalov se izvede v naslednjih fazah delovanja.

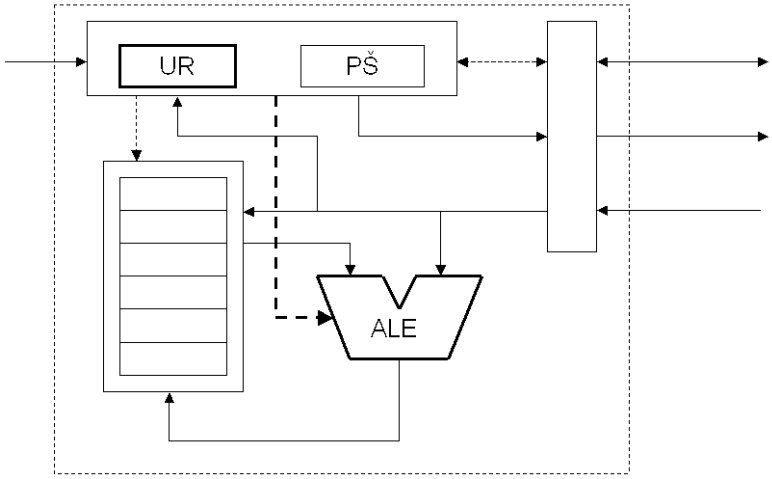


3. *Priprava parametrov*
   Če zahteva ukaz vhodne parametre (podatke), jih je potrebno predhodno pripraviti. Če se podatek nahaja v zunanjem pomnilniku, je potrebno na

naslovno vodilo pripeljati ustrezni naslov in s podatkovnega vodila prečitati njihovo vrednost. Če pa se parameter nahaja v registru, mora krmilna enota samo odpreti vrata med vodilom in enim izmed registrov. Čitanje zunanjih operandov običajno poteka poteka bistveno dalj časa od čitanja vrednosti registrov. Ta čas bi bil še daljši, če bi za izvedbo operacije potrebovali dva operanda iz zunanjega pomnilnika. V tem primeru bi potrebovali dodatni zadrževalni register (kot v primeru skupnega internega vodila za prenos podatkov), sama priprava parametrov pa bi zahtevala še en dodaten korak.

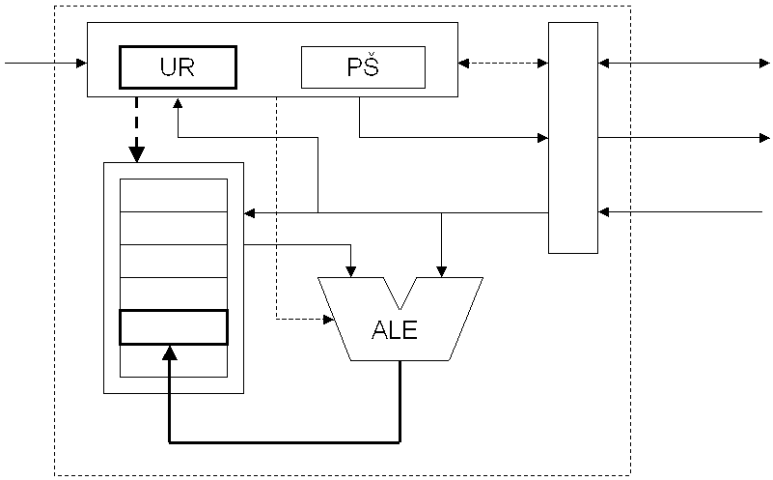


4. *Izvedba ukaza*
   Če gre za ukaze za obdelavo podatkov, za njegovo izvedbo običajno poskrbi ALE. Na vhodu sprejme do dva parametra in na izhodu generira rezultat. Tudi delovanje ALE je pod nadzorom ukaznega dekodirnika.



V določenih primerih ta izvajalna faza ni potrebna. Če smo želeli samo prenesti vsebino pomnilniške lokacije v enega izmed registrov, lahko to fazo izpustimo.

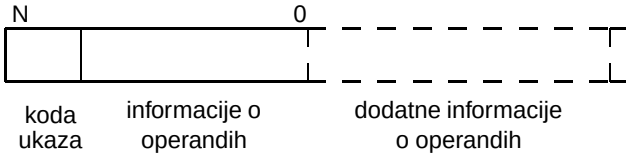
5. *Shranjevanje rezultatov*
   Vrednost rezultata je potrebno shraniti. Če je cilj operacije zunanji pomnilnik, je potrebno na naslovno vodilo pripeljati ustrezni naslov, na podatkovno vodilo pa prenesti izhodno vrednost ALE. Če je cilj operacije register, krmilna enota interno poveže izhod ALE z ustreznim registrom. V našem hipotetičnem modelu predpostavljamo, da je možno rezultat aritmetični/logične operacije shraniti samo v enega izmed registrov.



Fazi 3 in 5 sta odvisni od vrste ukaza in nista zmeraj prisotni. Da bi pospešili njihovo delovanje, so skoraj vsi sodobni mikroprocesorji organizirani tako, da se več ukazov znotraj mikroproesorja obdeluje sočasno. Ko se npr. izvaja dekodiranje enega ukaza, se sočasno že izvaja čitanje naslednjega ukaza. Podrobneje bomo o tej tehniki, imenovani cevenje (pipelining) in o drugih tehnikah za pohitritev delovanja mikroprocesorja govorili v 9. poglavju. Naš hipotetični model mikroprocesorja je zgrajen po filozofiji RISC mikroprocesorjev: vse aritmetično/logične operacije se izvajajo samo nad vsebino registrov; edina ukaza za povezavo z zunanjim podatkovnim pomnilnikom sta ukaza za shranjevanje in čitanje podatkov.

2.3.2 Oblika strojnih ukazov

Ukazi mikroprocesorja so zapisani v strojni obliki (binarno). Ukaz je lahko sestavljen iz ene besede ali več (razširitvenih) besed. Velikost besede je običajno enaka velikosti zunanjega podatkovnega vodila, lahko pa je tudi manjša.



V prvi besedi se običajno nahajata koda ukaza in informacija o operandih, v razširitvenih besedah pa so podane dodatne informacije o operandih (npr. naslov operanda v pomnilniku).

Krmilna enota najprej izlušči informacijo o operandih ukaza in jih po potrebi prenese v pomožne reistre ali na notranje vodilo procesorja, nato pa na osnovi kode ukaza sproži potrebno zaporedje signalov za njegovo izvedbo.

CISC mikroprocesorji uporabljajo ukaze različne dolžine (z različnim številom besed v operacijski kodi), RISC mikroprocesorji pa težijo k ukazom dolžine ene besede. S tem se poenostavi dekodiranje ukazov in izveča učinkovitost mikroprocesorja saj ne potrebujemo večkratnih posegov v programski pomnilnik. Nekatere mikroprocesorske arhitekture (npr. Intel) uporabljajo tudi t.i. predpone. To so posebne kode, ki jih navedemo pred ukazom in ki spremenijo njegov pomen. Na ta način so omogočili razširitev nabora ukazov pri novejših verzijah mikroprocesorjev in hkrati ohranili kompatibilnost strojne kode s starejšimi verzijami.

S povečanjem dolžine besede na 64 bitov se v sodobnih mikroprocesorskih arhitekturah pojavlja tudi možnost predstavitve več ukazov znotraj iste besede. V tem primeru se več ukazov mikroprocesorja dejansko izvede vzporedno. Seveda je zato potrebna tudi ustrezna mikroprocesorska arhitektura. Potrebujemo pa tudi poseben prevajalnik, saj ukazov ne moremo poljubno združevati. Veliko krat je potrebno, da se izvede najprej en ukaz in šele zatem se lahko izvede drugi.

2.3.3 Implementacija krmilne enote

Krmilna enota sodobnih mikroprocesorjev je lahko zasnovana na dva načina: s *kombinacijsko logiko* ali na *principu mikroprogramiranja*.

Večina zgodnjih mikroprocesorjev je uporabljala *kombinacijsko logiko*. Pri tem načinu se vsi ukazi interpretirajo z logičnimi vezji. Dekodirnik ukazov je sestavljen iz dekoderja, ki prepozna ukaz in ga razgradi v komponente, ter enkoderja, ki na njihovi osnovi generira krmilne signale za krmiljenje posameznih enot mikroprocesorja.

Mikroprocesorji iz skupine CISC so največkrat *mikroprogramirani*. Ta tehnika se je razvila iz želje po bogatem naboru kompleksnih ukazov, ki jih je bilo težko izvesti v diskretnih vezjih. Uporabili so osnovno idejo, ki je botrovala nastanku mikroprocesorjev: za izvedbo kompleksnih digitalnih vezij izdelamo univerzalno vezje, katerega obnašanje programiramo.

Dekodirnik ukazov deluje kot mikroprocesor v malem, saj vsebuje tako *mikroprogramsko kodo* kot tudi *mikroprogramski števec*. Njegova izvedba je zasnovana na preprosti kombinacijski logiki. Na ta način se en ukaz mikroprocesorja prevede v sekvenco mikro ukazov, ki so shranjeni kot *mikroprogram ali mikrokoda* v ROM-u (*firmware*). Mikro ukazi so precej enostavnejši od samih ukazov mikroprocesorja. Namesto operandov so deli mikro ukazov navodila za generiranje krmilnih signalov za ostale enote znotraj mikroprocesorja.

Tehnologija mikrokodiranih mikroprocesorjev omogoča kompleksnejše nabore ukazov in naslavljanj. Nabor ukazov lahko proizvajalec relativno preprosto spremeni ali razširi z zamenjavo vsebine mikro ROM-a brez spremembe kombinacijske logike. Ob revizijah pri izdaji nove verzije procesorjev preprosteje optimizirajo mikroprogramirano izvajanje ukazov. Tudi snovanje procesorja je preprostejše, hitrejše in ima jasno strukturo, posamezne komponente se lahko razvijajo vzporedno, napak je zaradi manjše kompleksnosti manj.

Obstajajo tudi *mikroprogramirljivi procesorji*, pri katerih je mikroprogram shranjen na posebnem programirljivem notranjem ali zunanjem pomnilniku in ga je mogoče spreminjati, graditi svoje ukaze ipd. Takšne tipe procesorjev danes le še redko srečujemo.

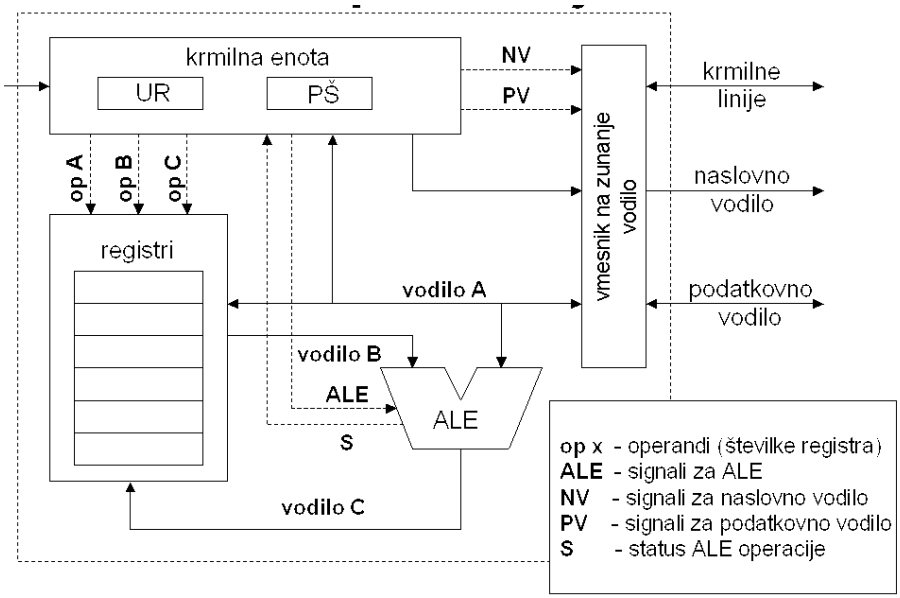
Slabost mikroprogramirane izvedbe izvajanja ukazov je v tem, da je delež krmilne enote na čipu v primerjavi z aktivnimi komponentami (ALE, registri) velik (v kompleksnih CISC mikroprocesorjih tudi več kot 60%), dekodiranje pa zaradi razkošnega nabora ukazov razmeroma počasno. Za izvršitev enega ukaza je potrebnih mnogo korakov. Zato so se v sodobnih mikroprocesorjih tipa RISC omejili na najpotrebnejše ukaze in tako spet omogočili izvedbo s kombinacijsko logiko, ki zavzema največ 10% površine čipa. Ob preprostejših optimirajočih prevajalnikih so s tem dosegli bistveno večje hitrosti izvajanja istih programov v višjem programskem jeziku.

V sodobnih mikroprocesorskih arhitekturah je krmilna enota lahko zelo kompleksna in organizirana tako, da se lahko več ukazov izvaja sočasno. O tem bomo podrobneje govorili v 9. poglavju.

2.3.4 Zgled delovanja hipotetičnega mikroprocesorja

V tem podpoglavju po predstavljen zgled možne izvedbe zgoraj opisanega hipotetičnega modela mikroprocesorja.

2.3.4.1 Arhitektura modela



Mikroprocesor sestavljajo splošno namenski registri (le-teh je 16), aritmetčno/logična enota, krmilna enota ter vmesnik za povezavo z zunanjim vodilom. Enote so medsebojno povezane preko treh vodil za prenos podatkov (vodila A,B in C), vodila za posredovanje naslova operanda v zunanjem pomnilniku (vodilo N), ter množice krmilnih linij (na sliki so označene s črtkano črto).

Preko vodila A se prenašajo podatki iz zunanjosti mikroprocesorja v bodisi krmilno enoto bodisi aritmetično/logično enoto. Po istem vodilu se lahko podatki iz enega izmed registrov prenesejo tudi navzven. Do katere pomnilniške lokacije želi mikroprocesor dostopati je določeno z naslovom podanim preko vodila N. Prenos podatkov v/iz mikroprocesorja je krmiljem s skupinama kontrolnih linij NV in PV. Prva skupina skrbi za odpiranje zunanjega naslovnega vodila, z drugo pa krmilimo odpiranje podatkovnega vodila in smer prenosa podatko na njem. Vodilo A skrbi tudi za prenos enega izmed operandov iz registrov v ALE.

Vodilo B služi za prenos drugega operanda v ALE in za posredovanje podatka pri posrednem naslavljanju, vodilo C pa za shranjevanje rezultata nazaj v enega izmed registrov. Kateri registri (in če sploh) nastopajo v operaciji je določeno preko krmilnih linij opA, opB in opC. Kot bo še prikazano, so te linije direktno

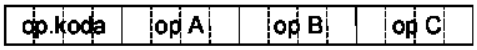
preslikane iz operacijske koda ukaza.

Kateri ukaz bo izvedla ALE je določeno z ALE krmilnimi signali. Na sliki so prikazane še krmilne linije označene z S. Te linije odražajo stanje zadnje aritmetično/logične operacije; npr. ali je bil rezultat zadnje operacije 0. So vhod v krmilno enoto in se uporabljajo pri pogojnih skokih. To so ukazi, ki prenesejo izvajanje programa na nek drug naslov v primeru, ko je izpolnjen določen pogoj.

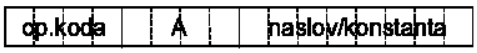
Na sliki je prikazan še signal za uro, ki daje krmilni enoti potreben takt za njeno delovanje.

2.3.4.2 Nabor ukazov

Nabor ukazov, ki ga takšen model podpira je prikazan na spodnji sliki. Prikazana je tudi zgradba ukazov mikroprocesorja. Uporabljen je model kjer so vsi ukazi enake dolžine (v našem primeru 32 bitov). Podrobneje bo o programiranju mikroprocesorja govora v ?? poglavju.



0000 - ADD a,b,c	$R_C = R_A + R_B$	
0001 - SUB a,b,c	$R_C = R_A - R_B$	
0010 - OR a,b,c	$R_C = R_A$ or R_B	
0011 - AND a,b,c	$R_C = R_A$ and R_B	
0100 - NEG a,b	$R_B = -R_A$	
0101 - NOT a,b	$R_B =$ not R_A	
0110 - MOV [a],b	$R_B = [R_A]$	[x] - vsebina pomnilnika
0111 - MOV a,[b]	$[R_B] = R_A$	

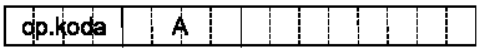


1000 - MOV naslov,A	$[n] = R_A$	MOV a,b ≡ OR a,a,b
1001 - MOV A,naslov	$R_A = [n]$	
1010 - MOV #konst,A	$R_A = k$	

Prva skupina ukazov je namenjena za izvedbo aritmetično/logičnih operacij. Vse operacije se izvedejo neposredno nad registri. Polja opA, opB in opC se preslikajo v isto imenske krmilne linije in krmilijo odpiranje ustreznih registrov. Potek priprave podatkov, izvedba ukaza in shranjevanje podatko se lahko izvede skoraj sočasno. V to skupino spadata tudi ukaza za posredni dostop do zunanjega

pomnilnika. Naslov operanda je določen z enim izmed registrov. Izvedba teh ukazov poteka v dveh fazah. V prvi fazi se vsebina enega izmed registrov prenese na naslovno vodilo. V drugi fazi se vsebina registra ali prenese na podatkovno vidilo ali pa se iz njega prečita.

1100	00 - JMP n	PŠ = n
1100	01 - JZ n	if ZERO then PŠ = n
1100	10 - JC n	if CARY then PŠ = n



1110 - JMP [a]	PŠ = R_A
1111 - MOV PC,a	R_A = PŠ

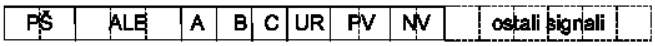
V drugo skupino ukazov spadata ukaza na neposredno nalaganje operandov iz/v zunanji pomnilnik. V tem primeru je naslov ukaza kar del ukazne kode. Dodan je še ukaz, ki v enega izmed registrov neposredno naloži konstantno vrednost zapisano v kodi ukaza.

V naslednji skupini se nahajajo ukazi za brezpogojne in pogojne skoke. Prvi povzroči brezpogojni prehod na drugo lokacijo v kodi programa, pri ostalih dveh pa se prehod izvede le, če je izpolnjen postavljen pogoj.

V zadnjo skupino ukazov spadata ukaza, ki omogočata dinamično spreminjanje vrednosti programskega števca (kot bomo videli v ?? poglavju, sta potrebna za izvedbo klicev podprogramov). Prvi izvede prenos vsebine programskega števca v enega izmed registrov, drugi pa prenos iz registra v programski števec sčemer se prenese izvajanje programa na neko drugo lokacijo.

2.3.4.3 Izvedba krmilne enote

Krmilna enota v podanem modelu mikroprocesorja deluje s pomočjo zaporednega izvajanja mikroukazov. V takšnem modelu se vsak ukaz mikroprocesorja preslika v zaporedje večih mikroukazov, ki se izvajajo po taktu zunanje ure ter generirajo ustrezne krmilne signale. Na spodnji sliki je prikazana sestava takšnega mikro ukaza. Določena polja neposredno ustrezajo krmilnim linijam, ki so opisana na sliki ??. Izjema so krmilne linije A, B in C, ki skupaj z ustreznimi biti operanda tvorijo krmilne linije opA, obB in opC, ter polje PŠ, ki krmili obnašanje programskega števca.



PŠ
- 00 - brez sprememb
- 01 - povečaj za 1
- 10 - naloži iz UR
- 11 - naloži iz vodila A

ALE
- 000 - brez sprememb
- 001 - C = A
- 010 - C = A+B
- 011 - C = A-B
- 100 - C = A or B
- 101 - C = A and B
- 110 - C = not A
- 111 - C = -A

A,B,C
- 0 - neaktivno
- 1 - dostop do registra (A in B branje, C pisanje)

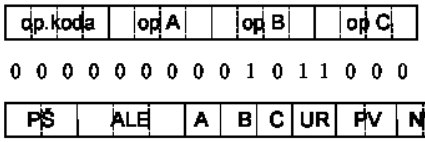
UR
- 0 - brez sprememb
- 1 - naloži iz vodila A

PV
- 00 - brez sprememb
- 01 - branje (na notranje vodilo A)
- 10 - pisanje (iz notranjega vodila A)

NV
- 00 - brez sprememb
- 01 - vsebina PŠ
- 10 - vsebina vodila A

številke registrov so določene z polji op A, op B in op C iz ukaza

Kako bi takšna kontrolna enota lahko delovala je prikazano na naslednjem zgledu operacije seštevanja dveh registrov:



0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 ADD R0,R5,R8

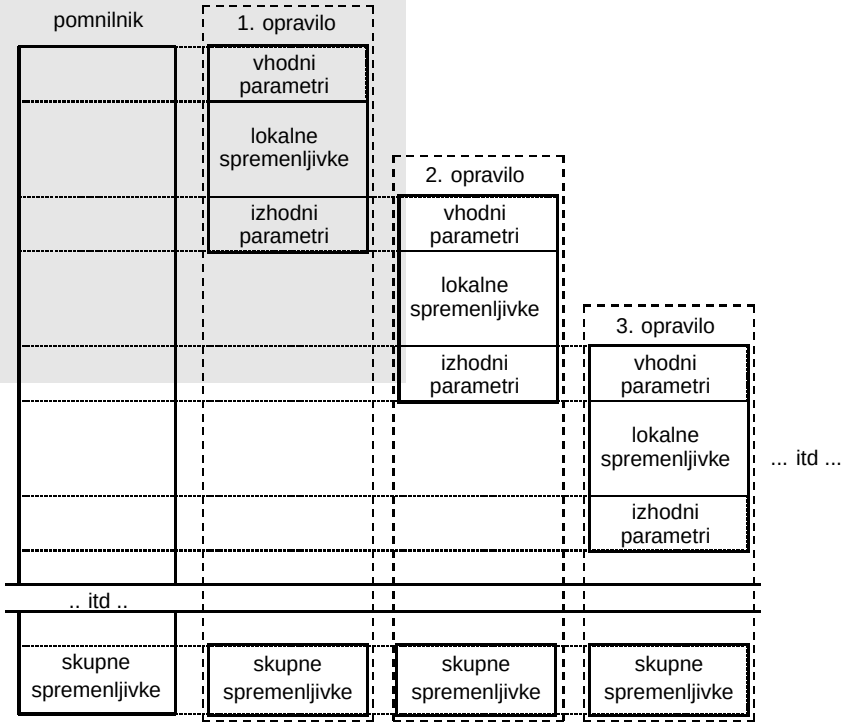
	PŠ	ALE	A	B	C	UR	PV	NV	ostali signali
1.	00	000	0	0	0	0	00	01	vsebina PŠ se prenese na naslovno vodilo
2.	01	000	0	0	0	1	01	00	vrednost is podatkovnega vodila se prepiše v UR, dekodiranje ukaza, PŠ = PŠ+1
3.	00	010	1	1	1	0	00	00	na vodilo A se prenese R0, na vodilo B se prenese R5, ALE izvede operacijo seštevanja, rezultat se shrani v R8
1.	00	000	0	0	0	0	00	01	vsebina PŠ se prenese na naslovno vodilo
2.	01	000	0	0	0	1	01	00	vrednost is podatkovnega vodila se prepiše v UR, dekodiranje ukaza, PŠ = PŠ+1

Prva dva koraka sta splošna in se morata izvesti ob vsakem ukazu. S prvim prenesemo vsebino programskega števca na naslovno vodilo. S tem sprožimo, da se koda naslednjega ukaza prenese na zunanje podatkovno vodilo. ALE enota ni aktivna, prav tako ne potrebujemo dostopa do registrov. V drugem koraku se vsebina iz podatkovnega vodila prenese v ukazni register. S tem se lahko začne dekodiranje ukaza. Glede na prve štiri bite ukaza se izvede skok na eno izmed 16 lokacij v mikrokodi (na primeru je to prikazano s črto). V drugem koraku se izvede še povečanje števca za ena. S tem se pripravimo na sprejem novega ukaza.

Kot rezultat dekodiranja ukaza se izvede tretji korak. V tem koraku se izvede več stvarih sočasno: na vodili A in B se preneseta vrednosti registrov R0 in R5, v ALE se sproži operacija seštevanja, vodilo C pa se poveže z registrom R8 (v njega se shrani rezultat).

Na koncu se ponovita prvi in drugi ukaz s katerima se bo izvedel naslednji ukaz programa.

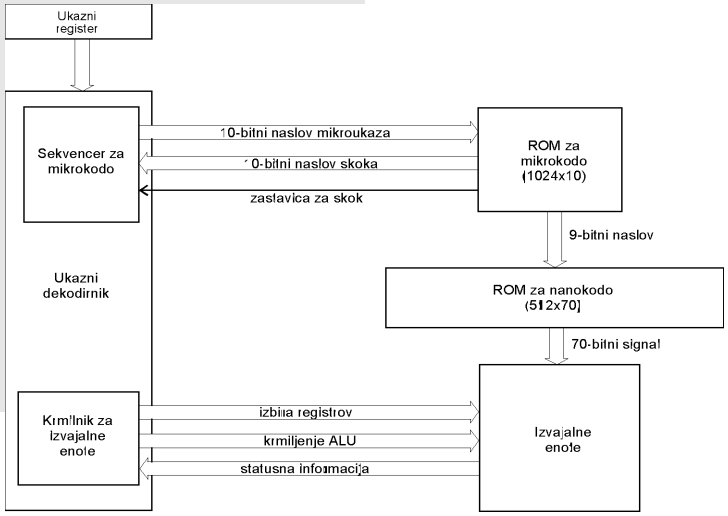
Pri RISC procesorjih so se temu izognili tako, da so namesto posameznih registrov uporabili registrski nabor (*register file*). V njem je večje število registrov. Vsakemu opravilu, ki teče na procesorju, se dodeli nekaj registrov, ki so mu dostopni (vidni) preko *okna*. To okno je sestavljeno iz treh delov, centralnega dela za lokalne spremenljivke, dela za vhodne parametre nad njim in dela za izhodne parametre pod njim. Ob klicu podprograma se izhodni del okna klicajočega programa prekrije z vhodnim delom okna klicanega programa in mu tako elegantno prenese parametre. Ob koncu klicanega podprograma se postopek ponovi v obratni smeri. Obstaja še okno, ki je vedno vidno vsem opravilom, in v katerem se hranijo skupne spremenljivke.



V prid organizaciji pomnilnika po oknih ter za njihovo dimenzioniranje govorijo rezultati raziskav, ki so jih izvajali razni avtorji (prevsem Knuth in Tanenbaum). Ti govorijo, da ima 92% procedur manj kot 6 lokalnih skalarnih spremenljivk, 30% pa celo nobene. 98% jih ima manj kot 6 vhodnih parametrov in 21% nobenega. Med operandi v Pascalskih in C programih je 20% konstant, 55% skalarnih spremenljivk in 25% polj ali drugih struktur. 80% skalarnih spremenljivk je lokalnih, medtem ko je 90% polj oz. struktur skupnih oz. globalnih. Iz tega sledi, da zadoščajo za lokalne spremenljivke ter za izmenjavo vhodnih oz. izhodnih parametrov že relativno majhna okna. Polja in večje strukture ja smiselno organizirati v skupnem pomnilniku.

Kot primer si poglejmo, kako je izvedena krmilna enota pri mikroprocesorju Motorola MC68000; ta je en od redkih mikroprocesorjev, katerih proizvajalec je objavil princip njenega delovanja. Pri snovanju so uporabili kompromis: uporabljajo dva nivoja mikrokode, od katerih se prvi izvaja vertikalno (*mikrokoda*), drugi pa horizontalo (*nanokoda*). Analiza je namreč pokazala, da je izvajanje makroukazov sestavljeno iz razmeroma majhnega števila elemetarnih akcij; te so zapisane v nanokodi, ki se razširja horizontalno in se zato hitro izvaja.
Vsak ukaz iz nabora ukazov tega mikroprocesorja je kodiran kot zaporedje mikroukazov. Mikroukaz lahko vsebuje naslov nanoukaza ali pa naslov mikroukaza, ki se naj naslednji izvede, kar si lahko predstavljamo kot neke vrste skok v *"mikro–subrutine"*.
Nanoukazi so dolgi 70 bitov; iz njih dobimo z dekodiranjem približno 180 fizičnih krmilnih signalov. V nano ROM-u je 280 nanoukazov, kar zahteva 9-bitni naslov. Mikroukazov je 640, za kar zadostuje 10-bitni naslov. Vsebujejo lahko 9-bitni naslov nano- (format 2) ali 10-bitni naslov naslednjega mikroukaza (format 1), zato njihov format sestavlja 10 bitov (glej sliko spodaj). V primeru, ko gre za naslov nano ukaza (format 2), ostane na razpolago en bit, ki se uporablja za indikacijo, kaj je vsebina naslednjega mikroukaza, nano- ali naslednji mikroukaz.

Arhitektura krmilne enote mikroprocesorja Motorola MC68000 je prikazana na spodnji sliki.



Ukaz mikroprocesorja se iz ukaznega registra prenese v dekodirnik. Tam se del, ki določa registre in delovanje ALE, izloči in posreduje nadzorniku enot za izvajanje operacij, ostanek pa uporabi sekvencer za mikrokodo. Ta nadzoruje izvajanje mikroukazov, tako, da pošilja ROM-u naslove mikroukazov, ki se naj izvedejo. Najprej pošlje začetni naslov mikroprograma, nato pa praviloma zaporedne naslove naslednjih mikroukazov; če nek mikroukaz ne vsebuje naslova nanoukaza in torej zahteva skok na drugi mikroprogram, se ciljni naslov posreduje nazaj sekvencerju. Mikroukazi, ki vsebujejo naslove nanoukazov, pošljejo le-te v ROM za nanokodo. Iz nanoukazov se generira 70-bitni signal, ki upravlja z enotami za izvajanje operacij. Mikroprogram se zaključi s posebnim mikroukazom.

Poglavje 3

Programiranje mikroprocesorjev

V prejšnjih poglavju je bilo podrobneje opisano notranje delovanje mikroprocesorja v tem poglavju pa si bomo podrobneje pogledali programski vidik uporabe mikroprocesorja. Najprej do predstavljeni tipi podatkov s katerimi lahko mikroprocesor operira in kako so ti podatki fizično predstavljeni. V nadaljevanju bomo podrobneje pregledali ukaze, načine naslavljanja in druge značilnosti, ki so pomembne za programiranje. Programiranje mikroprocesorjev neposredno v strojni obliki je praktično neuporabno. Zato za izdelavo programov vedno uporabimo primernejše oblike zapisa. Za programiranje mikroprocesorjev na osnovnem nivoju je najprimernejši zbirni jezik saj omogoča neposredni dostop do vseh ukazov in funkcij mikroprocesorja. Programiranje v zbirnem jeziku pa je že pri nekoliko večjih programih zamudno. V primeru ko ne potrebujemo najoptimalnejše kode je veliko ugodneje, če uporabimo enega izmed višjih programskih jezikov. Sodobni prevajalniki že lahko generirajo kodo, ki je blizu optimalne pričemer pa je potrebni trud in čas za izdelavo programa bistveno krajši.

3.1 Predstavitev podatkov v računalniku in njihovo kodiranje

Običajna naloga programov je, da manipulirajo s podatki na takšen ali drugačen način pri čemer predstavljajo podatki različne fizikalne ali matematične vrednosti.

Mikroprocesorji lahko te vrednosti obdelujejo le na binaren način (kot množico ničel in enic). Zato je te vrednosti na nek način potrebno predstaviti v mikroprocesorju razumljivi obliki.

Osnovni tipi podatkov v mikroprocesorju so običajno cela števila. Kako velika števila lahko predstavimo je običajno pogojeni z dolžino njegove notranje besede oz. velikostjo njegovih registrov. Binarno kodirana cela števila so glede na potrebno dolžino lahko predstavljena z zlogi (byte), besedami (dva zloga, word), dolgimi besedami (dve besedi, long word) in četvornimi besedami (osem zlogov, quad word). Lahko so predznačena ali nepredznačena, pri čemer se negativna predznačena števila predstavljajo z dvojiškim komplementom svoje absolutne vrednosti. Negativne vrednosti tako predstavljenih števil na oko prepoznamo po najvišjem bitu, ki je enak 1.

Močnejši mikroprocesorji imajo vgrajene procesne enote za realna števila, predstavljena s plavajočo vejico (floating point), za obdelavo katerih so bili prej potrebni aritmetični ko-procesorji. Ti podatki so običajno kodirani na način, ki ga predpisuje IEEE 32-bit FP standard P754. Način njihovega kodiranja je podan na sliki (xx).

Slika za Single in Double (s formulami)

Eksponent števila je podan v dvojiški obliki z odmikom (127 pri enojni oz. 1023 pri dvojni natančnosti), kar pomeni, da dejansko vrednost eksponenta dobimo, če vrednosti v kodiranem številu odštejemo 127 oz. 1023. Mantisa je podana kot absolutna vrednost z eksplicitnim predznakom. V obeh prvih primerih je matisa normirana na vrednost 1.xxx, kar pomeni, da je največji bit vedno 1; ker bi torej pomenil redundantno informacijo, so ga izpustili in tako pridobili na natančnosti. Spodnja formula podaja vrednost števila, predstavljenega v formatu s plavajočo vejico z enojno natančnostjo:

$$x = (-1)^s * 2^{e-127} * 1.f$$

pri čemer je s predznak, e vrednost eksponenta, f pa vrednost mantise; pri dvojni natančnosti se namesto 127 od eksponenta odšteje 1023.

Posebnost kodiranja realnih števil je, da ima poleg regularnih še iregularne vrednosti, s katerimi lahko opišemo rezultate neveljavnih operacij (NaN, not-a-number)

ter prekoračitve obsega (predznačena neskončna vrednost). Primer za prve je rezultat izraza ∞/∞, primer za druge pa deljenje z nič. Vrednost za $\pm\infty$ je predstavljena z največjim mogočim ekponentom in mantiso, enako nič; predznak mantise pove, ali gra za plus ali minus neskončno. Enak, največji eksponent in neničelna mantisa podaja neobstoječo vrednost (NaN).

Za neposredno predstavitev in obdelavo desetiških števil je bil vpeljan posebni BCD kod (Binary-Coded Decimals). V tem zapisu so desetiška števila 0 – 9 so predstavljena s svojimi šestnajstiškimi ekvivalenti, vrednosti A – F pa so prepovedane; kombinacije 1010 – 1111 so torej neuporabljene. BCD vrednosti so lahko nepakirane ali pakirane (eno ali dve BCD števili v zlogu). Starejši mikroprocesorji so imeli ukaze, ki so omogočali neposredne operacije nad takšnimi tipi podatkov. V sodobnih mikroprocesorjih takšno predstavitev podatkov srečamo le redko.

Razen števil v mikroprocesorjih pogosto nastopajo tudi znaki, ki so kodirani po različnih standardih. V enostavnejših programih se največrat uporablja kodiranje po standardu ASCII (American Standard Code for Information Interchange). V osnovnem naboru je 128 znakov, za kar je potrebnih 7 bitov. Z razširitvijo osnovnega nabora s posebnimi znaki za mednarodne abecede in grafičnimi simboli so se naslovi razširili na osem bitov, kar se tudi sklada z dolžino zlogov.

Večina mikroprocesorjev podpira tudi delo s posameznimi biti ali množicami bitov.

Druge tipe podatkov običajno prevedemo na enega izmed zgoraj naštetih tipov. Tako lahko npr. časovne vrednosti predstavimo v obliki celih števil, ki povedo koliko milisekund je preteklo od polnoči. Operacije nad časovnimi vrednostmi se tako prevedejo na operacije s celimi števili.

3.2 Programski model

Programski model mikoprocesorja je slika, ki jo mora poznati programer pri izdelavi programa v zbirnem jeziku. Programski model predstavlja množica registrov do katerih ima programer neposredni ali posredni dostop, abstraktna slika pomnilnika v katerm se nahajajo program in podatki, nabor ukazov mikroprocesorja in načini naslavljanja operandov. K programskemu modelu prištevamo tudi sliko vhodno/izhodnih vmesnikov in njihovih registrov, ki bodo podrobneje opisani ka-

sneje.

3.2.1 Registri

Registri so hitre pomnilne celice znotraj mikroprocesorja, ki se uporabljajo za izvedbo ukazov mikroprocesorja. Število in funkcije posameznih registrov se glede na tip mikroprocesorja lahko razlikujejo lahko pa jih v grobem lahko razdelimo na podatkovne in naslovne. Prvi se uporabljajo pretežno za izvajanje matematičnih, logičnih in drugih operacij nad operandi, služijo pa tudi kot začasna shramba. Naslovni registri hranijo pretežno naslove pomnilniških lokacij posameznih operandov, ki v operacijah nastopajo. V večini sodobnih mekroprocesorjev je delitev med podatkovne in naslovne registre v veliki meri že zabrisana, saj se lahko nek register uporabi na oba načina.

Za nekatere registre je značilno, da imajo vnaprej določen namen. Takim registrom pravimo, da so namenski. Ostale registre označujemo kot splošne. Najznačilneji primeri nameskih registrov, ki jih srečamo v mikroprocesorjih so:

1. *Programski števec (program counter - PC)*
   Ta register smo že omenili pri opisu krmilne enote. Njegova naloga je, da kaže na (naslavlja) naslednji ukaz, ki ga mora mikroprocesor izvesti. Ob normalnem izvajanju programa se njegova vrednost linearno povečuje. Njegova vrednost se spremeni le ob skokih ter ob klicih podprogramov oz. vrnitvah iz njih.

2. *Kazalec sklada (stack pointer - SP*
   Vlogo kazalca sklada in uporabo sklada pri mikroprocesorjih bomo podrobneje spoznali nekoliko kasneje. Preko tega registra so implementirani mehanizmi klicev podrogramov, prenos parametrov v podprograme ipd.

3. *Statusni register (status register - SR)*
   V izvajanje programa spadajo tudi odločitve (npr. stavek if). Za implementacijo teh ukazov na nivoju mikroprocesorja običajno poskrbi statusni register. Ta je sestavljen iz množice bitnih vrednosti (zastavic - flags), ki odražajo rezultat zadnje izvedene operacije mikroprocesorja. Najpogostejše zastavice, ki jih srečamo pri skoraj vseh mikroprocesorjih so:

   Slika SR z zastavicami 4.7 iz Ribariča

Slika seštevanja dveh binarnih števil in prikaz zastavic

Z - zero: se postavin na vrednost 1, če je bil rezultat prejšnje operacije enak 0

C - carry/borrow: postavi se v primeru, ko vrednost rezultata preseže osnono velikost operanda (npr., ko pride do prenosa pri operaciji seštevanja ali do izposoje pri operaciji odštevanja)

V - overflow: postavi se ob prekoračitvi formata za predznačeno število. Pri operacijah nad predznačenimi števili prekoračitve ne moremo opazovati z bitom C (carry), ker je območje predznačenih števil manjše (najvišji bit predstavlja predznak).

N - negative: je postavljen, če je bil rezultat prejšnjega ukaza nad predznačenimi števili negativen (torej je imel postavljen najvišji bit)

Če želimo na primer ugotoviti ali sta dva operanda enaka, ju najprej medsebojno odštejemo nato pa opazujemo vrednost zastavice Z. Če sta enaka, bo rezultat odštevanja enak 0 in zastavica Z se bo postavila na 1.

Statusni register se uporablja tudi za prikaz trenutnega stanja (režima delovanja) mikroprocesorja. Takšne so na primer zastavice povezane s prekinitvami o čemer bo govora v poglavju o izjemah (**??**).

Sivi okvirček za skladovni procesor brez registrov

3.2.2 Organizacija pomnilnika

Število registrov znotraj mikroprocesorja je običajno premajhno, da bi lahko v njih hranili vse spremenljivke, ki nastopajo v programu. Zato je potrebno uporabiti pomnilnik. Pomnilnik je običajno organiziran v obliki zaporedja lokacij enake širine kot je širina registrov mikroprocesorja. Vsak zlog v pomnilniku je označen z enolično številko (naslovom), ki si linearno sledijo.

Slika pomnilnika

V splošnem programer razmišlja o pomnilniku le na abstrakten način, dejanski naslovi spremenljivk ga običajno ne zanimajo. V določenih primerih pa je ugodno, če lahko programer vpliva na lokacijo določene spremenljivke. Pri 32 bitnih mikroprocesorjev je dostop do dolge besede na lokaciji deljivi s štiri bistveno hitrejši kot v primeru ko naslov ni poravnan.

3.2.3 Nabor ukazov

Kako bomo z mikroprocesorjem implementirali nek program, je seveda odvisno od nabora ukazov, ki jih le-ta lahko izvaja. Podobno kot pri registrih se tudi tukaj nabor ukazov od procesorja do procesorja bistveno razlikuje. V grobem je nabor ukazov odvisen od vrste mikroprocesorja: CISC procesorji imajo veliko množico komplesnih ukazov, ki se lahko izvajajo tudi dalj časa (več urinih ciklov mikroprocesorja), RISC procesorji pa imajo manjšo množico enostavnejših ukazov, ki pa se izvajajo izredno hitro (v povprečji znotraj enega urinega cikla mikroprocesorja).

Operacijam, ki se izvajajo znotraj procesorja lahko razdelimo na:

- enočlenske operacije z unarnimi (monadnimi) operatorji nad enim samim operandom ($A := \ op\ B$), in
- dvočlenske operacije z diadnimi operatorji ($A := B\ op\ C$).

Glede na to ločimo eno-, dvo- in tro-naslovne ukaze. Pri slednjih podamo oba operanda in mesto kam se naj shrani rezultat. Ker so statistike pokazale, da so zelo pogosti izrazi tipa $A := A\ op\ B$, je smiselo uvesti dvo-naslovni format, pri katerem je dogovorjeno, da je en od operandov obenem vhod in, po operaciji shramba za rezultat. Ta format tudi skrajša dolžino ukaza in je največkrat uporabljen v mikroprocesorjih.

Starejši mikroprocesorji so izvajali dvočlenske operacije nad akumulatorji, ki so predstavljali podobno kot pri dvo-naslovnem formatu vir in shrambo za rezultat, in nad enim dodatnim operandom ($acc := acc\ op\ B$). Ker je bil akumulator eden, ali pa je bila njegova identifikacija implicitno skrita v operacijski kodi, je edini operand podatek B, zato je ta format eno-naslovni.

3.2.3.1 Skupine ukazov

Glede na namembnost lahko nabor ukazov razdelimo v naslednje skupine:

1. *Ukazi za premikanje*
   To so ukazi, ki običajno predstavljajo največji delež programa. Služijo za premikanje operandov iz pomnilnika v registre, iz registra v register ipd. Nekateri mikroprocesorji omogočajo, da se ob oremiku izvede tudi konverzija podatkov (npr. razširitev vrednosti iz besede na dolgo besedo).

2. *Aritmetični ukazi*
   V to skupino spadajo ukazi za matematične operacije (seštevanje, odštevanje, množenje, deljenje ter sprememba predznaka). Nekateri mikroprocesorji lahko te operacije izvajajo tako nad celoštevilčnimi vrednostmo kot rudi nad realnimi števili. Nekateri mikroprocesorji omogočajo tudi izvedbo komplesnejših matematičnih funkcij (kotne funkcije, logaritmi, ipd.).

3. *Logični ukazi*
   V to skupino spadajo ukazi za logične operacije (in, ali, izključni ali (seštevanje po modulu 2) in negacija).

   Slika tabelice logičnih operacij.

4. *Ukazi pomikanja in rotiranja*
   Posebna skupina ukazov za obdelavo podatkov so ukazi za premik ali rotacijo bitov znotraj besede, ki jih lahko prištevamo k aritmetičnim ali logičnim operacijam. Gre za pomike bitov v operandih v levo ali v desno. Pri premikih (shift) se v izpraznjeno mesto vpisujejo ničle ali se podvaja zadnji bit, tisti, ki izpade iz besede, gre v bit C (bit prenosa). Pri rotacijah se v izpraznjeno mesto vpisuje staro stanje prenosa C, ki se nato prekrije z bitom, ki je izpadel iz besede. Premiki se uporabljajo za paralelno/serijsko pretvorbo, dostop do posameznih bitov v besedi in za množenje ali deljenje z mnogokratniom 2 (premik za mesto v levo/desno je ekvivalenten moženju z bazo številskega sistema).

   Slika pomika in rotiranja.

5. *Ukazi za manipulacijo z biti*
   Večina mikroprocesorjev omogočajo operacije nad posameznimi biti zno-

traj operanda. Določen bit lahko postavimo, zbrišemo, zamenjamo njegovo vrednost ali pa ga testiramo.

6. *Ukazi za nadzor poteka programa*
   Ti ukazi se uporabljajo za določanje zaporedja obdelave ukazov. Sem spadajo ukazi za skoke, ki so lahko brezpogojni (absolutni) ali pogojni glede na stanje statusnega registra. Pomembna sta tudi ukaza za klic podprograma in povratek iz njega.

7. *Sistemski ukazi*
   Ti ukazi posegajo v stanje sistema oz. ne spadajo v nobeno od gornjih skupin. To so npr. ukaz STOP, ki zaustavi mikroprocesor ter ukaz NOP (no operation), ki razen, da porabi določeno število urinih ciklov, ne naredi nič.

8. *Operacije nad nizi, bloki in sestavljeni ukazi*
   V nekaterih CISC mikroprocesorjih najdemo ukaze, ki opravljajo kompleksne funkcije nad nizi, premikajo cele bloke podatkov in izvajajo operacije za podporo ukazom višjega programskega jezika za tvorbo zank, alternativ ipd.

3.2.4 Načini naslavljanja operandov

Načini naslavljanja nam povedo, kje se nahajajo operandi, nad katerimi naj se izvedejo specificirani ukazi. Ločimo *izvorne* in *ciljne* naslove. Procesor dobi operand z izvornega naslova, rezultat pa shrani na ciljni naslov. Operandi posameznih ukazov so lahko konstante, lahko so registrih mikroprocesorja ali pa določene pomnilniške lokacije. Pri sklicu na registre zadostuje, da navedemo njihovo ime (številko). Ko pa se operandi nahajajo v pomnilniku imamo na razpolago več možnosti za njihovo identifikacijo.

Pri CISC procesorjih (v nasprotju z RISC procesorji) je trend, da z elegantnimi načini naslavljanja čim bolj pospešimo izvajanje programov oziroma olajšamo njihovo pisanje. Kompleksnejši načini naslavljanja dovoljujejo določanje naslovov ob izvajanju programa namesto ob njegovem pisanju oziroma prevajanju, kar omogoča uporabo kompleksnejših podatkovnih struktur. Pri RISC procesorjih se večina operacij izvaja nad registri, tako da moramo najprej operande pripeljati v posamezne registre, šele nato lahko nad njimi izvedemo operacijo.

Večina sodobnih mikroprocesorjev podpira naslednje načine naslavljanja:

- *takojšnje* (immediate): operand je kot konstanta naveden v samem ukazu ali v raširitveni besedi, ki mu sledi;
- *registrsko neposredno* (register direct): operand se nahaja v enem od registrov, ki je podan z ukazom;
- *registrsko posredno* (register indirect): naslov operanda se nahaja v z ukazom specificiranem naslovnem registru; Pri tem načinu naslavljanja je mogoče specificirati tudi *odmik* in/ali *indeks*. *Odmik* je pozitivno ali negativno absolutno število, ki se prišteje k vrednosti podanega naslovnega registra,*indeks* pa je odmik, ki je shranjen v enem izmed preostalih registrov in se prav tako prišteje končmenu naslovu. Ta način naslavljanja se uporablja pri delu s tabelami. V osnovni register vpišemo začetni naslov tabele v pomnilniku z indeksom pa dinamično izbiramo, kateri element želimo nasloviti.
- *absolutno* (absolute): v ukazu je podan neposredni naslov operanda, ki se nahaja v pomnilniku;
- *relativno* (relative): naslov operanda se izračuna relativno glede na trenutno vsebino programskega števca. Ta način naslavljanja omogoča pisanje t.i. *relokatibilnih* programov[1].

3.3 Sklad v mikroprocesorju

Sklad je podatkovna struktura, ki je zelo pomembna za delovanje mikroprocesorjev. Je struktura *LIFO*[2], kar pomeni, da je podatek, ki je zadnji šel na sklad, na vrhu te strukture in bo prvi prečitan iz nje.

[1] programov, ki jih lahko naložimo in izvajamo kjerkoli v pomnilniku
[2] Last In, First Out

3.3.1 Realizacija sklada pri mikroprocesorjih

Sklad je pri mikroprocesorjih običajno realiziran kot del pomnilnika za podatke. Pomembno vlogo pri tem ima kazalec sklada (stack pointer - SP), ki kaže na zadnjo vpisano lokacijo na sklad. Skoraj vedno je sklad organiziran tako, da poteka od višjih pomnilniških lokacij proti nižjim. Ob vpisu na sklad se vrednost kazalca sklada najprej zmanjša, nato pa se na lokacijo na katero kaže vpiše nov podatek. Pri čitanju s sklada se najprej prečita vrednost na katero trenutno kaže kazalec sklada nato pa se vrednost kazalca ustrezno poveča.

Slika zapisovanja in čitanja s sklada

Sklad vsebuje ključne informacije za delovanje mikroprocesorja. Če se njegova vsebina pokvari ali če izgubimo naslov v kazalcu sklada, se delovanje nepreklicno poruši.

3.3.2 Uporaba sklada

Osnovni namen sklada je, da hrani vrnitvene naslove ob klicu podprogramov. Ob klicu podprograma se trenutna vrednost programskega števca prenese na sklad, nato pa se izvede skok na začetek kode podprograma. Na tan način si mikroprocesor zapomni, od kod je bil podprogram dejansko poklican.

Slika JSR

Ob vrnitvi iz podprograma se vrednost iz sklada ponovno naloži v programski števec in izvajanje se nadaljuje na mestu tik za klicem.

Slika RET

Skladovna organizacija je potrebna, ker se klici podprogramov lahko gnezdijo, kar pomeni, da se lahko znotraj enega podprograma kliče drugi podprogram itd. V tem primeru bo na skadu zapisano več povratnih naslovov in sicer v obratnem vrstnem redu klicanja.

Sklad se lahko uporabi tudi za prenose parametrov v podprograme. Tik pred klicem podprograma se vrednosti parametrov zapišejo na sklad nato pa se izvede klic. Podprogram se na te parametre lahko sklicuje preko posrednega naslavljanja glede na kazalec sklada. Po vrnitvi iz podprograma se vrednost kazalca sklada

popravi na vrednost, ki je bila pred zapisom paramtrov.

Na skladu se običajno hranijo tudi lokalne spremenljivke, ki so deklarirane znotraj podprogramov. Ker je v določenih primerih možno nek podprogram klicati znova še preden se je tekoči klic končal (npr. ob rekurzivnem klicu), uporaba lokalnih spremenljivk na fiksnih lokacijah ne pride v poštev. Vsak novi klic podporgrama bi namreč pokvaril vrednosti teh spremenljivk iz prejšnega klica. V tem primeru si pomagamo tako, da na začetku podprograma rezerviramo del sklada za te spremenljivke (kazalec sklada ustrezno premaknemo - zmanjšamo za skupno velikost lokalnih spremenljivk). Tik pred izhodom iz podprograma pa ta prostor sprostimo (kazalec premaknemo na prejšnjo vrednost). Ob vgnezdenem ponovnem klicu bo ostal nabor lokalnih spremenljivk nedotaknjen. Aktiven je vedno le najvišji nabor spremenljivk. Tako zasnovane programe imenujemo ponovno izvedljive (“reentrant”).

Potreba po uporabi sklada se pojavi tudi ob vseh ostalih prekinitvah normalnega izvajanja programov, imenovanih izjeme. Razlika je v tem, da se v tem primeru shrani ne le povratni naslov, temveč tudi druga informacija, ki je potrebna za nemoteno nadaljevanje prvotnega prekinjenega programa (podrobnejša razlaga je podana v poglavju **??**).

3.4 Zbirni jezik

Edini razumljiv program za procesor je program v *strojnem jeziku*. Ta je sestavljen iz binarno kodiranih zlogov in je za programerja zelo težko čitljiv, pisanje programov v njem pa bi pomenilo naporno delo in zelo veliko verjetnost napak. Zato so bili računalniki že zgodaj (1969) opremljeni z *zbirnim jezikom*. Definira ga proizvajalec procesorja; običajno je enoten za celo družino mikroprocesorjev, čeprav se strojni ukazi lahko razlikujejo. Obstajali so poskusi, da bi standardizirali zbirni jezik za vse mikroprocesorje, vendar se niso obnesli.

3.4.1 Splošna oblika ukazov v zbirniku

Ukazi v zbirnem jeziku je podobna zgradbi ukaza mikroprocesorja v strojni kodi t.j. koda ukaza in seznam operandov. Zato imajo tudi ukazi v zbirnem jeziku

podobno obliko:

labela operacijska_koda operand$_1$ operand$_2$ .. operand$_n$ komentar

Dejanski format, število naslovov in velikost posameznih polj ukaza so različni od mikroprocesorja do mikroprocesorja. Operacijska koda ukaza v zbirnem jeziku je *mnemonik*[3], ki se skupaj z operandi in glede na način naslavljanja direktno preslika v strojno kodo. *Labela* je označba, ki si jo zbirnik[4] zapomni skupaj z naslovom prve lokacije ukaza in na katero se lahko kasneje sklicujemo, ne da bi morali poznati pravi naslov. Labela je neobvezno polje in se pojavi le, kadar je potrebna, na primer za definiranje pomožnih lokacij, za označevanje skokov v programu ipd. *Komentar* je lahko ločen od ostalega dela ukaza z *delimiterjem*, podaja informacijo, potrebno za lažje razumevanje pomena ukaza oziroma za dokumentiranje programa. Komentar se lahko pojavlja tudi kot edino polje v vrstici; takrat se mora kot prvi pojaviti nek rezerviran znak, običajno $*$, podpičje, klicaj ipd.

V zbirnem jeziku ločimo dve skupini ukazov, *izvedljive ukaze* in *smernice za zbirnik*.

3.4.1.1 Izvedljivi ukazi

Izvedljivi ukazi so tisti ukazi, ki se neposredno preslikajo v mikroprocesorski ukaz v strojnem jeziku.

3.4.1.2 Smernice za zbirnik (Assembler Directives)

Smernice za zbirnik ali *psevdoukazi* ne rezultirajo v izvedljivi strojni kodi, temveč služijo kot ukazi prevajalniku. Pišemo jih v polju za operacijske kode v zgornji shemi. Uporabljamo jih za *definiranje simbolov, prireditev pomnilnika, strukturiranje programa, formatiranje, krmiljenje poteka prevajanja ter vklop oziroma izklop vhodno-izhodnih enot* ipd..

[3] Mnemonika: [*iz gr. mneme spomin*] spretnost in nauk o urjenju spomina, ki temelji na zakonih o asociaciji idej (Verbinc, Slovar tujk); v imenu ukaza je torej skrita asociacija o pomenu.

[4] prevajalnik za zbirni jezik, *(assembler)*

Primer smernic za *definiranje simbolov* je smernica EQU (*Equate, izenači)*, ki logičnemu simbolu, navedenem kot labela, priredi neko konstantno vrednost, npr.:

```
STEVEC EQU 100
```

Primer *prireditve pomnilnika* je smernica npr. DS (*Define Storage*), ki rezervira specificirano število sledečih lokacij pomnilnika), ali pa smernica ORG (*Origin, izvor*), ki običajno na začetku programa postavi naslov, na katerega se bo prevajal program.

Večina zbirnikov zmore tudi razširjanje *makrojev*. Kadar se del programa pogosto ponavlja, ga je ugodno zapisati kot *makro* z uporabo ustreznih zbirniških smernic za *strukturiranje programa*. Tak makro se ob klicu vključi v program. Bistvena razlika med podprogrami in makroji je v tem, da so makroji le olajšava pri pisanju programov, njihova koda se v celoti vključi v izvorni program. Klic podprograma se prevede kot skok na del programa, ki je naložen nekje ločeno in le enkrat. Pri uporabi podprogramov štedimo pomnilniški prostor na račun časa, ki je potreben za skok na rutino. Pri klicu makroja ni skoka, ker se ta nahaja na istem mestu, vendar vsak klic pomeni razširjanje kode in s tem porabo pomnilnika.

Za *krmiljenje poteka* imamo pri večini zbirnikov na voljo posebno opcijo, pogojno prevajanje[5]. S smernico IF testiramo stanje neke logične spremenljivke, ki krmili, ali se bo del izvornega programa med IF in ENDIF vključil v prevajanje ali ne. Na ta način lahko npr. z istim izvornim programom delamo implementacijsko odvisne objektne verzije.

3.5 Višji programski jeziki

Značilnost ukazov v zbirnem jeziku je, da se enoumno preslikajo v strojne, kar pomeni, da ima programer možnost in dolžnost eksplicitnega snovanja programov na najnižjem nivoju. Nasprotno temu je programiranje v *višjem programskem jeziku*, kjer ga praviloma ne zanima, kako se bo stavek iz njegovega programa prevedel v strojni. Za to pretvorbo poskrbi prevajalnik, ki generira bolj ali manj optimalno kodo.

[5] Conditional Assembly

Slika primer prevedbe IF stavka

Iz tega izhajajo razlike pri programiranju v zbirnem in višjem programskem jeziku: v zbirnem jeziku so programi bolj optimalni, lepše izkoriščajo možnosti procesorja, generirajo krajši strojni program in so hitrejši. Programer v zbirnem jeziku preprosto dosega periferne vmesnike in vse drugo, kar mu nudi mikroprocesor in mikroračunalnik.

Kadar programira v višjem programskem jeziku, pa ima na drugi strani na voljo orodje, ki bistveno poenostavi programiranje. Programi so (po vrsticah) krajši, bistveno bolj čitljivi in jih je mnogo lažje vzdrževati. Zaradi vsega tega so manj občutljivi na napake in jih je preprosteje verificirati. Tudi produktivnost programerjev (količina programa v časovni enoti) je bistveno večja.

Ko se odločamo o snovanju aplikacije na osnovi mikroračunalnika, se moramo smiselno odločiti, v kakšnem jeziku bomo programirali. Programiranje v zbirnem jeziku se iz zgoraj navedenih razlogov vedno bolj opušča. Smiselne so kvečjemu kompromisne odločitve, kjer najbolj kritične dele programiramo v zbirnem večino programa pa v višjem programskem jeziku, oziroma, ko izvedemo ročno optimizacijo programov.

3.6 Orodja in pripomočki za razvoj programov

Učinkovita izdelava programov za mikroprocesorje je povezana z uporabo ustreznih razvojnih orodij. Pri tem imamo na razpolago več možnosti:

- *Zbirniki in prevajalniki*
  O zbirniku in prevajalniku smo govorili že v prejšnjem podpoglavju. Oba izvorno kod programa preslikata v izvedljivo strojno kodo mikroprocesorja. Manjši miroprocesorji in mikroprocesorski sistemi nimajo dovolj kapacitet, da bi lahko prevajanje potekalo neposredno na njih. Seveda tudi za na novo zgrajene sisteme nimamo na razpolago ustreznih prevajalnikov. Zato običajno programe razvijamo na drugih (večjih) sistemih. V tem primeru se prevajalnik izvaja na enem sistemu, kodo, ki jo generira pa se bo izvajala na drugem. V tem primeru govorimo o t.i. *navzkrižnih zbirnikih* oz. *navzkrišnih prevajalnikih* (cross assembler, cross compiler).

- *Simulatorji in emulatorji*
  Simulatorji simulirao delovanje ciljnega mikroračunalniškega sistema in omogoča, da programe zan razvijamo in preizkušamo še preden zgradimo ciljni sistem. Emulatorji so naprednejša oblika simulatorjev, ki jih lahko vključimo neposredno v ciljni sistem (v podnožje za mikroprocesor) . Po eni strani lahko tako opazujemo delovanje znotraj mikroprocesorja, po drugi strani pa emulator emulira obnašanje mikroprocesorja na strojnem nivoju tako, da lahko opazujemo tudi obnašanje končnega sistema.

- *Očiščevalniki (debugers)*
  Očiščevalniki so namenjeni odkrivanju logičnih napak v programu. Običajno delujejo tako, da omogočajo izvajanja programa korak po korak pri čemer lahko opazujemo stanja posameznih registrov in/ali pomnilniških lokacij ter drugih elementov sistema. Nastavljamo lahko prekinitvene točke, vplivamo na vrednosti posameznih vrednosti v sistemu ipd. Očiščevalniki so običajno vključeni neposredno v razvojno orodje (simulator) ali pa se (v primitivni obliki) nahajajo v programski kodi ciljnega sistema.

Za sodobna razvojna orodja je značilno, da so vse zgoraj naštete funkcije povezane v celoto.

Poglavje 4

Prenos podatkov

Pri snovanju mikroračunalnikov se v glavnem ukvarjamo s povezovanjem univerzalnih komponent (mikroprocesorja, pomnilnikov in perifernih enot) v celoto z vodili, po katerih se prenašajo podatki v najširšem pomenu besede. Gre za informacije o naslovih in podatkih ter za krmilne signale, ki upravljajo delovanje računalnika.

V tem poglavju bomo obdelali več nivojev prenosa, od fizičnih signalov, protokolov prenosa, do višjih funkcij in primerov vodil. V poglavju 6 bomo analizirali primer zgradbe preprostega hipotetičnega mikroračunalnika.

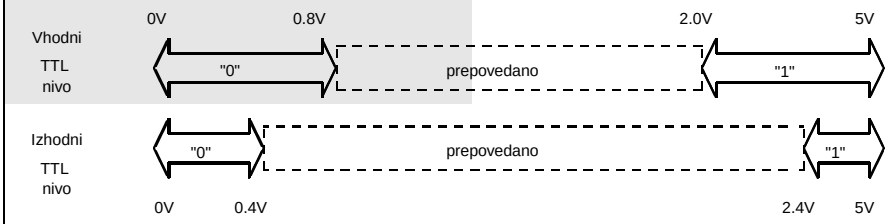
4.1 Signali

Komunikacija med posameznimi enotami v mikroračunalniku poteka preko *signalov*. Ti so pri mikroprocesorjih običajno fizično realizirani kot električne veličine, največkrat napetost, in se prenašajo preko *povezovalnih linij*.

Za prenašanje podatkov se lahko uporabijo tudi druge fizikalne velčine; zelo pogost je optični prenos podatkov na večje razdalje. Ta način ima prednosti, kot so hitrost prenosa in neobčutljivost na motnje zaradi električnih pojavov.

Napetostni signali imajo svoje predpisane nivoje, ki določajo, katerim napetostim pripadata logični vrednosti 0 in 1.

Mejni napetostni nivoji v tradicionalni TTL tehnologiji na primer določajo logično 0 za napetosti med 0 V in 0,4 V in logično 1 med 2,4 V in 5V za izhodne signale. Med 0,4 V in 2,4 V se signal ne sme nahajati; prehod skozi to področje mora biti čim hitrejši.
Pri vhodnih signalih je dovoljene več tolerance: ničla seže do 0,8 V, enica je že od 2 V naprej. Vzrok je v tem, da se signal pri potovanju po liniji "pokvari", pridobi motnje, pade nekaj napetosti.



(Pozor: merilo za prikaz napetosti je zelo popačeno!)

V zadnjem času se zaradi tehnološkega napredka lahko uporabljajo precej nižji napetostni nivoji, ki omogočajo višje hitrosti prehodov iz enega nivoja v drugega in pri katerih se pri konstantnem toku sprošča manjša energija v obliki toplote, ki jo moramo odvesti s čipov.

Signale določajo *tri lastnosti*: *pomen, smer in stanje*

- *pomen*: opis vpliva signala na delovanje sistema (naslovni, podatkovni, urini, prekinitveni, statusni in drugi signali); podrobni opisi inčasovni diagrami so podani v priročnikih uporabljenih mikroprocesorjev;
- *smer*: vhodni, izhodni, dvosmerni signali, glede na smer pretoka informacije; smer signalov določamo s stališča mikroprocesorja: mikroprocesor generira izhodne signale;
- *stanje*: signal je lahko v visokem (*"1"*) ali nizkem (*"0"*) stanju. Če je v visokem stanju signal pravilen (veljaven, "true"), govorimo o pozitivni logiki, v nasprotnem primeru pa o negativni. Nekateri signali so lahko tudi v posebnem stanju visoke upornosti (visoke impedance, HiZ), kar pomeni, da je signal odklopljen od ostalega vezja. To stanje ni namenjeno za prenos informacij, temveč za priključitev več virov signalov na isto linijo in se realizira s posebno tehnologijo izhodnih vezij.

Signali se med komponantami mikroračunalnika prenašajo po *linijah*, ki predstavljajo fizično povezavo med njimi.

4.2 Vodila

Skupina linij, ki tvorijo funkcionalno celoto, je *vodilo*. Običajno govorimo o treh skupinah linij: o *podatkovnem*, *naslovnem* in *krmilnem vodilu*, pri čemer je krmilno vodilo ime za skupino linij, ki jim je skupna nadzorna funkcija, ne pa tudi pomen posamezne linije. Kot *vodilo* v širšem pomenu besede običajno smatramo celoto vseh treh vodil. Fizično predstavlja vodilo skupina žic, linij, povezav na tiskanem vezju itd..

Odločitev o tem, kakšno bo vodilo v nekem mikroprocesorskem sistemu, je en od prvih korakov pri njegovem snovanju. Izbira vodila je odvisna od uporabljenega mikroprocesorja, zahtevanih zmogljivosti, vezanih na prenos podatkov, načina strežbe prekinitev, dodeljevanja prenosnega medija različnim enotam, ki lahko zahtevajo prenos podatkov in drugih značilnosti sistema. Pomembno je, da je zmogljivost vodila pravilno izbrana; vodila s premajhno prepustnostjo bodo kmalu povzročila ozko grlo pri zmogljivosti sistema, prerazkošna vodila pa lahko zahtevajo preveč dodatnih vezij, kar sistem podraži in ga naredi preveč kompleksnega.

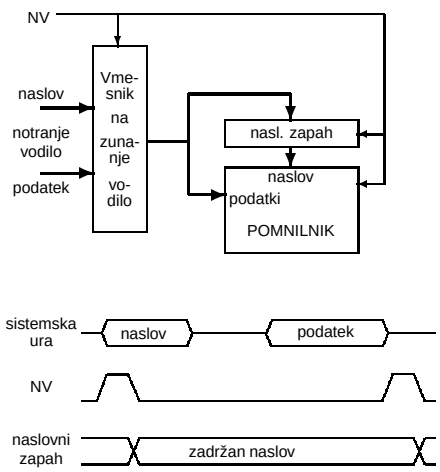
Vodila delimo na *multipleksirana* (deljena) in *nemultipleksirana*. Pri multipleksiranih imamo na nožicah v enem trenutku ene, v drugem druge signale, kar je sinhronizirano s krmilnimi linijami, povezanimi s sistemsko uro.

Običajno so multipleksirani naslovni in podatkovni signali, kar je pogojeno z naravo prenosa podatkov: pobudnik prenosa najprej pošlje naslov lokacije; medtem, ko se vrši dekodiranje naslova, se na vodilu pripravijo in nato prenesejo podatki. Multipleksirane signale najdemo na primer v Intelovi družini procesorjev in v dinamičnih RAM pomnilnikih.

Prednost multipleksiranih signalov je v tem, da zahtevajo manj priključkov, kar pomeni manjša in zato cenejša vodila, in čipe, ki zavzemajo tudi manj prostora na tiskanem vezju. Nemultipleksirani so preprostejši, omogočajo hitrejši dostop in enostavnejši razvoj sistemov.

Vodila delimo tudi na *dostopna in nedostopna*. Nedostopna so skrita v povezavah na tiskanem ali celo integriranem vezju. Primerna so za sisteme, pri katerih niso predvidene spremembe in so običajno vgrajeni v druge naprave. Dostopna vodila so običajno izvedena kot niz konektorjev, v katere lahko vtikamo različne mikroračunalniške module in s tem gradimo ustrezno konfiguracijo sistema. Njihova

Na sliki je prikazan mikroprocesorski vmesnik za multipleksirano vodilo. Ločena podatkovni in naslovni vmesnik ter programski števec so preko skupnega vmesnika in multiplekserja vezani na multipleksirano zunanje vodilo. Krmilni signal NV (Naslov-na-Vodilu) ob svoji aktivni fronti pove, kdaj se na vodilu nahaja naslov. Zunanja enota (npr. pomnilnik) ob negativni (padajoči) fronti NV tega takrat zadrži v zapahu (latch) in ga uporabi za dekodiranje in izbiro podatka. Ko je podatek na razpolago, se lahko vodilo uporabi za njegov prenos. Časovno dogajanje na multipleksiranem vodilu je prikazano na časovnem diagramu. Signal NV naj bo aktiven ob svoji negativni fronti.



prednost je torej fleksibilnost, slabost pa, da je treba na vsakem modulu izdelati vmesnik za vodilo, kar sistem podraži in poveča.

Fizično so vodila običajno izdelana kot povezave na tiskanem vezju. Možne so tudi optične povezave, katerih uporaba pa je iz praktičnih razlogov omejena na serijska vodila.

V modelu mikroprocesorja na sliki v poglavju 1.1 so na vodilo priključeni procesor, pomnilnik in periferni vmesniki. V splošnem je lahko v mikroračunalniku tudi več procesorjev oziroma drugih naprav, ki lahko zahtevajo prenos podatkov.

Na vodilo so lahko priključeni trije tipi enot:

- *aktivne enote (gospodarji)*: enote, ki lahko zahtevajo dostop do medija za prenos podatkov. Tipični predstavnik tega tipa je mikroprocesor.
- *pasivne enote (sužnji)*: enote, ki se lahko le odzivajo na zahteve po prenosu podatkov in ga same ne morejo zahtevati. Tipični predstavnik je pomnilnik.
- *kombinirane enote*: enote, ki so lahko aktivne ali pasivne, vendar ne istočasno. Njihova primarna funkcija je pasivna; ko jih neka aktivna enota izbere in naloži neko nalogo, postanejo aktivne, dokler ne izpeljejo zahtevane ak-

cije, nakar se vrnejo v pasivno stanje. Tipični predstavnik je DMA krmilnik (glej poglavje 7.3).

4.2.1 Mehanske in električne značilnosti vodil

Vodila so običajno izvedena kot matične plošče (motherboard) s konektorji. Istoležni priključki na vseh konektorjih so običajno povezani med seboj (razen v primeru marjetične verige, pozicijskega kodiranja ipd.) Ker je priključkov veliko na razmeroma ozkem prostoru (tipično 96 vzporednih linij), so linije blizu ena poleg druge, kar povzroča razne električne probleme (presluh - signal z ene linije se zaradi indukcije čuti na sosednji- , kapacitivne izgube ipd.)

Snovalci se trudijo, da bi bile linije čim tanjše in s tem razmik med njimi čim večji. Presluh zmanjšamo tudi tako, da linije obremenimo z upori, s čemer majhne inducirane tokove odvedemo proti masi; seveda to po drugi strani zahteva močnejše gonilnike koristnih signalov. Presluhe preprečujemo še z večplastno tehnologijo tiskanih vezij, kjer sosednje linije porazdelimo na različne plasti na obeh straneh ploščice ter tako povečamo razdalje med njimi; nekatere plasti v celoti uporabimo za maso in napajanje, s čemer druge linije dodatno oklopimo; slaba stran te rešitve je dodatno povečanje kapacitivnih izgub.
Posebne probleme povzročajo pojavi zaradi visokih frekvenc signalov in njihove pravokotne oblike, ki vključujejo višje harmonske komponente osnovne frekvence; pojavljajo se odbita in stojna valovanja ipd., proti čemer ukrepamo z aktivnimi obremenitvami linij.

Konektorje, ki so prispajkani na matično ploščo in v katere vtikamo module, delimo na enodelne in dvodelne. *Enodelni konektor* se nahaja na osnovni plošči; vanj vtikamo ploščice, ki imajo na robu izdelane in običajno trdo pozlačene kontakte ali *robne konektorje*, kot jih najdemo v osebnih računalnikih (PC). Zaradi tehnologije izdelave ploščic je gostota kontaktov na robnih konektorjih omejena.

Kadar potrebujemo večjo gostoto kontaktov, uporabimo *dvodelne konektorje*, pri katerih imamo na vsaki ploščici po en del konektorja, (moški in ženski), ki se ob vtikanju sestavita. Dvodelni konektorji so tudi zanesljivejši in zaščiteni pred vplivi okolja (prah, vlaga itd.), vendar so dražji. Pomembna in standardizirana je tudi razdalja med konektorji na osnovni plošči. Čim manjša je, tem manjše so dimenzije celotnega sistema, vendar je s tem oteženo hlajenje in zmanjšana dovoljena višina elementov na vtični ploščici.

Izvedba konektorjev je zelo pomembna za zanesljivost delovanja mikroračunalnika. Peresa v kontaktih morajo biti dovolj močna, da ustvarijo zanesljiv kontakt in prebijejo tudi morebitni sloj oblog iz nečistoč in oksidov, ki se lahko ustvari na kontaktih. Po drugi strani pa močna peresa (pri tipično 96 kontaktih na konektorju) zahtevajo razmeroma veliko silo pri vtikanju in izvlačenju ploščic in zaradi tega posebne mehanske izvedbe in pripomočke.
Posebne linije in po več priključkov na konektorjih je namenjeno napajalnim napetostim in masi; nekatere kartice potrebujejo za svoje delovanje razmeroma veliko energije, napajalni tok je velik in zahteva ustrezne povezave. Priključki za maso na konektorjih so včasih podaljšani, s čemer zagotovimo, da se bodo zadnji razklenili; tako omogočimo, da lahko plošče priključujemo in odklapljamo z vodila med delovanjem.

4.2.2 Oddajniki in sprejemniki električnih signalov

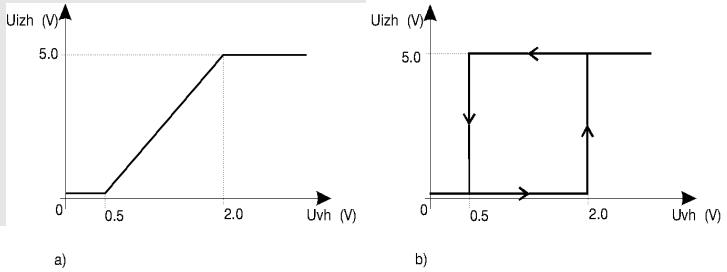
Čeprav izgleda električni prenos signalov po vodilu preprost, je treba v realnih razmerah paziti na vrsto problemov. Nekaj jih je bilo že opisanih v gornjem poglavju; večinoma izhajajo iz dejstva, da so signali relativno visokih frekvenc.

Na svoji poti od izvora pa do vhoda naslednjega elementa se lahko oblika signalov popači: napetost se zniža zaradi potrošnje energije na linijah, pojavijo se motnje zaradi induciranih napetosti in presluha med linijami itd. Da bi bili s tem povezani problemi čim manjši, uporabljamo za pošiljanje in sprejem signalov posebna namenska vezja, *sprejemnike in oddajnike*.

Oddajniki so tokovni ojačevalniki, ki zagotavljajo ustrezne signale tudi pri obremenitvah z več TTL bremeni in pri daljših linijah. Zagotavljajo tudi ločitev med ploščami in vodilom. Običajno imajo oddajniki možnost postaviti izhode v stanje visoke upornosti, s čemer modul navidezno odklopijo od vodila. Drugi oddajniki imajo izhode izvedene v tehniki z odprtim kolektorjem, kar omogoča, da lahko njihove izhode vežemo na isto linijo; na ta način lahko realiziramo ožičeno “ali” operacijo nad njimi.

Sprejemniki imajo za nalogo določiti logični nivo podatka na liniji glede na TTL nivoje in s tem zmanjšati čas zadrževanja signala v nedovoljenem napetostnem območju vhodnega signala.

V praktičnih izvedbah pogosto najdemo sprejemnike in oddajnike kombinirane v enem čipu. Ti elementi imajo dodatne vhode, ki povejo smer prenosa podatkov in postavljajo izhode v stanje visoke upornosti.



Korekcijo vhodnih sigmalov lahko opravijo na dva načina, ki sta prikazana na gornji sliki. V prvem primeru je ob vhodnih signalih pod 0,5 V izhodni vedno 0 V, pri vhodnih nad 2 V je izhodni 5 V, vmes pa je prepovedano prehodno območje, kjer signal hitro narašča. V drugem primeru je prikazan primerjalnik s histerezo (Schmitov prožilnik), kjer je izhodni signal 0 V, dokler vhodni ne doseže vrednosti za visok nivo (2 V), oziroma ostane na visokem, dokler vhodni ne pride do 0,5 V.

4.3 Načini prenosa podatkov

Podatke lahko po vodilih prenašamo na paralelni ali na serijski način, zato vodila delimo na *paralelna in serijska*. Pri paralelnih so skupine signalov (npr. podatki oz. naslovi) istočasno prisotne na linijah. Pri serijskih so signali časovno multipleksirani, običajno na eni sami podatkovni liniji, prenos pa je sinhroniziran z dodatno sinhronizacijsko linijo. Prednost serijskega vodila je v tem, da je primerno za prenos podatkov na večjo razdaljo, da zahteva manj vmesnikov in ga je mogoče izvesti z optičnimi vlakni, vendar je zaradi sprotnega časovnega multipleksiranja in demultipleksiranja signalov prenos podatkov bistveno počasnejši kot pri paralelnih vodilih.

Enote, ki sodelujejo v operacijah znotraj mikroračunalnika, običajno delujejo avtonomno in neodvisno druga od druge; med seboj so povezane le preko vodila za prenos podatkov. Za pravilni prenos podatkov morata biti partnerja v komunikaciji, torej gospodar in suženj, oziroma sprejemnik in oddajnik, *časovno usklajena*. Naslovljena naprava mora torej vedeti, kdaj je naslov pravilen, prejemnik lahko prečita podatke s podatkovnih linij, ko je njihova veljavnost zagotovljena, oddajnik jih ne sme umakniti, dokler niso prečitani, itd. Z ozirom na način časovnega usklajevanja med komunicirajočima enotama ločimo dva načina prenosa podatkov, *sinhroni* in *asinhroni* način.

V nadaljevanju podajamo splošni opis sinhronega in asinhronega protokola pri paralelnem prenosu podatkov. Kot primere navajamo resnične protokole preprostih (in zastarelih) Motorolinih mikroprocesorjev M6800 in MC68000, ki pa so razeroma čisti in dobro ilustrirajo tukaj opisane principe.

Prikazan je tudi serijski prenos podatkov s sinhronim protokolom na primeru Phillipsovega vodila I^2C, ki ga je privzela Motorola pod imenom M-bus.

Na koncu podajamo popolni primer zgradbe standardnega vodila VME, ki je najbolj razširjeno na področju mikroprocesorskih sistemov v procesnem vodenju. V poglavju 9 pa prikazujemo še osnovne značilnosti vodil, ki jih nahajamo v osebnih računalnikih.

4.3.1 Protokoli prenosa podatkov po paralelnem sistemskem vodilu

Ko procesor želi prenašati podatke, naslovi želenega partnerja v komunikaciji s tem, da na linije naslovnega vodila postavi pripadajoče bite njegovega naslova. S krmilnimi linijami poda ostale podatke (smer prenosa, velikost podatka in druge.) Pasivna enota, ki prepozna svoj naslov, se ustrezno odzove ter sprejme ali pošlje podatke prek linij podatkovnega vodila. Ta postopek je časovno usklajen z usklajevalnimi signali, ki se bistveno razlikujejo med sinhronim in asinhronim načinom.

4.3.1.1 Sinhroni način prenosa podatkov

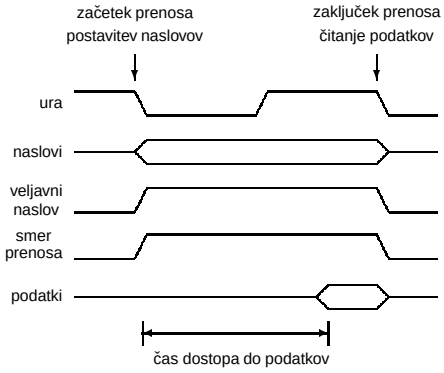
Sinhrona komunikacija je preprostejši način prenosa podatkov in zahteva manj vezij. Osnova delovanja je *skupna ura*, ki se prenaša po vodilu in na katero so sinhronizirane komunicirajoče naprave, ali *lokalne ure*, ki so sinhronizirane prek zanesljivega sinhronizacijskega mehanizma. Rešitev s porazdeljenimi urami je bolj zahtevna, pa tudi bolj odporna na napake, saj v primeru izpada skupne ure izpade ves sistem, v primeru porazdeljenih ur pa le nekatere komponente. Zaradi tega se zmanjšajo njegove performanse, delni sistem pa lahko preživi.

Urin signal ima dve fronti, rastočo in padajočo, ki običajno določata vse aktivnosti pri prenosu podatkov. Za delovanje potrebujejo vse enote določen minimalni čas, ki je podan v priročnikih. Periodo urinega signala določimo tako, da zmore tudi

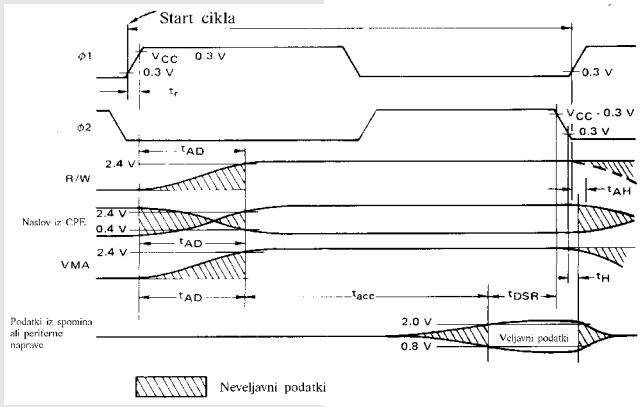
najpočasnejša enota prenesti podatke.

Sinhrona komunikacija zahteva natančno definirane začetke in konce vseh operacij na vodilu. Dobesedno bi to pomenilo, da bi moral biti naslov pravilen natanko vso prvo polperiodo cikla prenosa podatkov, vso drugo polperiodo pa bi morali biti veljavni podatki. Ti bi se praviloma torej morali pridobiti pred začetkom te polperiode, oz. med tem, ko je veljaven naslov.

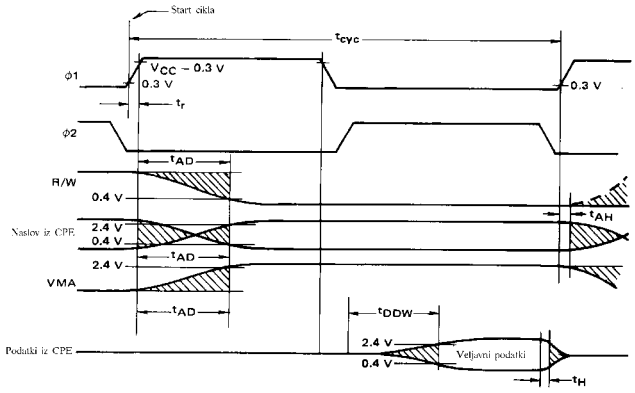
Običajno je to prestrogo in ni smiselno, saj je na primer dovolj, da so naslovi prisotni ob začetku prenosa in ni posebej pomembno, kdaj jih procesor umakne, čeprav jih pomnilnik več ne rabi. Podatke pomnilnik prečita vselej ob koncu cikla in jih ne potrebujemo že ob začetku druge polperiode, zato si lahko vzamemo čas za njihovo pridobitev od takrat, ko so naslovi veljavni, pa skoraj do konca druge polperiode. Cel cikel, določen z urino periodo, je zato lahko krajši, frekvenca višja in vezja preprostejša. Takšen način komunikacije se imenuje *semisinhroni* in je bil uporabljen pri večini preprostejših osembitnih mikroprocesorjev.



Na gornji sliki je prikazan poenostavljen princip semisinhronega protokola. Kadar želi procesor komunicirati z eno od pasivnih enot, ob ustrezni fronti urinega signala postavi njen naslov na naslovne linije. Z visokim signalom *smer prenosa* (po dogovoru) pove, da želi čitati podatke, z nizkim bi jih pošiljal. S signalom *veljavni naslov* pove, da v resnici gre za prenos podatkov. Nekateri procesorjevi cikli so namreč namenjeni njegovim notranjim opravilom; čeprav se tudi takrat naslovne linije nahajajo v nekem stanju, ne vsebujejo informacije o naslovu in ne bi bilo zaželeno, da bi se pomnilnik takrat odzival.



Kot primer si oglejmo semisinhroni protokol čitanja podatkov pri enem najstarejših mikroprocesorjev, Motorola *M6800*, ki je prikazan na zgornji sliki.
Za delovanje je uporabljal *M6800* dva urina signala, ϕ_1 in ϕ_2, ki sta bila v protifazi in nikoli istočasno v visokem stanju. Za prenos podatka med procesorjem in periferno enoto je bila potrebna ena urina perioda.
Cikel se začne ob naraščajoči fronti ϕ_1. Procesor postavi na vodilo naslov partnerja, smer komunikacije ($R/\overline{W}$), in signal *VMA*, ki pove, da so naslovi veljavni (značilnost Motorolinih procesorjev je kombinirani signal $R/\overline{W}$; ker sta obe stanji aktivni, je potrebno s posebno linijo *VMA* povedati, da v resnici gre za prenos podatkov).
Te linije se morajo ustaliti najkasneje po t_{AD} (Address Delay). Pasivne naprave tedaj začnejo dekodiranje naslova in tista, ki se prepozna, poišče zahtevani podatek ter ga postavi na vodilo. To se zgodi po času t_{ACC} (Access Time). Ta trenutek mora biti po naraščajoči periodi oziroma t_{DSR}(Data Set-up on Read, čas, potreben, da se signali stabilizirajo) pred padajočo fronto ϕ_2. Padajoča fronta ϕ_2 povzroči čitanje podatkov z vodila in umik vseh ostalih signalov ter s tem zaključek cikla.



Pri pisanju je situacija podobna, le da tokrat procesor sam postavi podatke na vodilo z zakasnitvijo t_{DDW} (Data Delay on Write) za signalom *DBE*(Data Bus Enable), na katerega je običajno pripeljan ϕ_2. Periferija mora prečitati podatke v času, ko je *DBE* aktiven.

Pasivna naprava v ustreznem trenutku, definiranim z *uro* in signalom *veljavni naslov*, prepozna svoj naslov in priskrbi želene podatke, za kar potrebuje nekajčasa. Ta mora biti krajši kot čas do zaključka periode; do takrat morajo podatki biti na razpolago in ob naslednji fronti urinega signala jih procesor prečita z vodila.

Če se zgodi, da pridejo podatki na vodilo prepozno, procesor tega ne more ugotoviti. V tem primeru bo prečital s podatkovnih linij naključno vrednost.

Frekvenca urinega signala je zato postavljena tako, da zmorejo komunicirati tudi najpočasnejše enote. To pa pomeni, da bodo morale hitre enote preostali čas po tem, ko opravijo svojo nalogo, čakati. Zato se trudimo v sistemu s sinhronim prenosom podatkov uporabiti enote s primerljivimi odzivnimi časi.

Včasih to ob snovanju sistema ni mogoče, pač pa se hitrejše enote pojavijo kasneje, ko je sistem že zasnovan. Če želimo tedaj počasne enote nadomestiti s hitrejšimi, je potrebno spremeniti ves sistem. Sinhroni protokol je torej nefleksibilen glede na konfiguracijo enot.

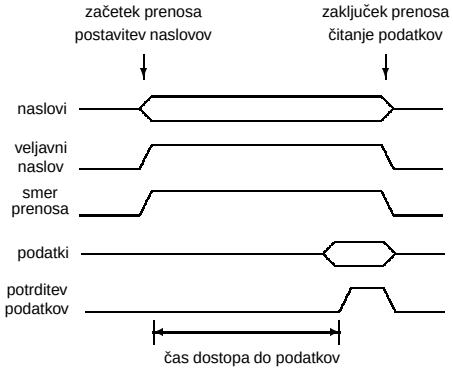
Nesposobnost prilagajati hitrost prenosa hitrosti naprav so včasih skušali omiliti s *podaljševanjem ure*. V ta namen so generator urinih signalov izdelali tako, da je bilo mogoče periodo ure podaljšati z zunanjim signalom. Ko je dekodirnik naslovov ugotovil, da je naslovljena počasna naprava, je generiral signal za podaljševanje, ki je zahteval od generatorja ure, da podaljša cikel prenosa podatkov za toliko, da bodo podatki pripravljeni. Cikla ni bilo mogoče podaljševati v nedogled, kar je pogojeno z dinamično izvedbo registrov, katerih osveževanje je vezano na urino frekvenco.

4.3.1.2 Asinhroni način prenosa podatkov

Sodobni procesorji skoraj brez izjeme uporabljajo asinhroni način prenosa podatkov. Pri tem načinu je odpravljena nefleksibilnost pri različnih hitrostih enot. Enota pri asinhronem načinu odgovori v najhitrejšem možnem času in istočasno potrdi veljavnost podatkov brez čakanja na vnaprej predvideni trenutek za sinhronizacijo. Cena za to je kompleksnejša aparaturna izvedba.

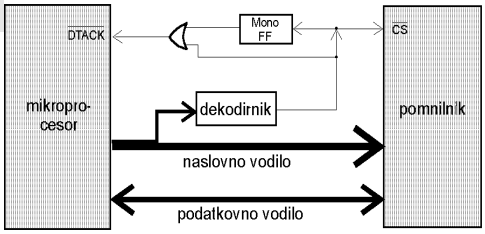
Spodnja slika prikazuje poenostavljen princip asinhronega protokola za prenos podatkov. Mikroprocesor v poljubnem trenutku postavi na vodilo naslov podatka in smer prenosa ter potrdi veljavnost teh signalov. Ob signalu "veljavni naslov"podrejene enote preverijo naslov in tista, ki se prepozna, začne pridobivati podatke. Ko so na voljo, jih da na vodilo in to potrdi z ustreznim signalom. Na

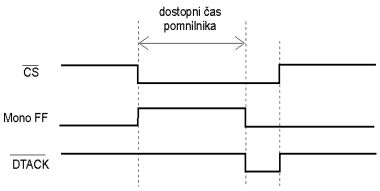
tega čaka mikroprocesor, prečita podatke ter umakne naslov in krmilne signale, kar pomeni, da je cikel uspešno zaključen. Kot posledica tega tudi podrejena enota umakne podatke in potrditveni signal in s tem sprosti vodilo za naslednji cikel.



Iz opisanega sledi, da procesor čaka, dokler ne dobi signala, da so na razpolago veljavni podatki. V primeru, da se je zgodila napaka (npr. izpad pomnilnika), tega signala sploh ne bo dobil. V tem primeru bi sistem "obvisel" oz. čakal v neskončnost. Da bi to preprečili, moramo sami poskrbeti za časovni nadzor prenosa podatkov.

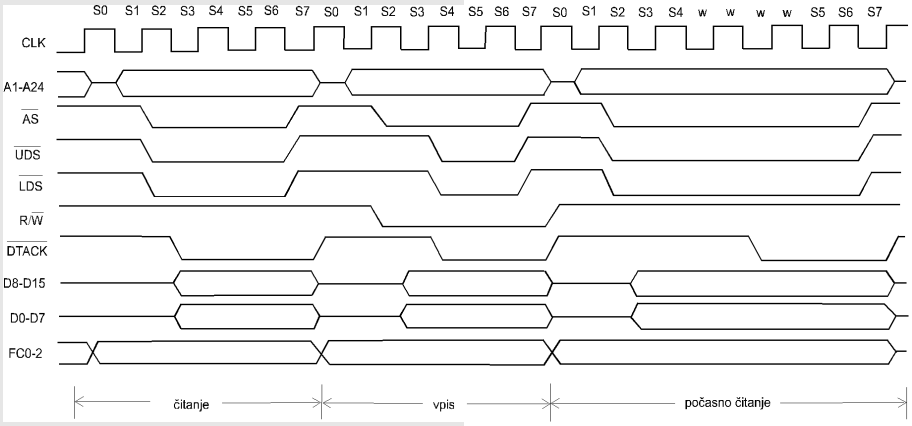
Na sliki je prikazan primer vezja za generiranje signala za potrditev podatkov ($\overline{DTACK}$), ki se nahaja na podrejeni enoti (npr. pomnilniku). Naslovni dekodirnik prepozna naslov in generira signal $\overline{CS}$, s katerim izbere pomnilnik in povzroči, da le ta začne iskati podatke. Z istim signalom tudi sproži monostabilni flip-flop, katerega perioda je nastavljena na čas, ki je enak dostopnemu času pomnilnika. Na izhodu ALI vrat dobimo v času od izteka impulza monostabilnega multivibratorja pa do konca signala $\overline{CS}$ negativni signal $\overline{DTACK}$; njegova začetna (negativna) fronta je znak mikroprocesorju, da je protokol s strani pomnilnika gotov. Ko umakne naslove, se konča tudi $\overline{CS}$ in posledično še $\overline{DTACK}$.



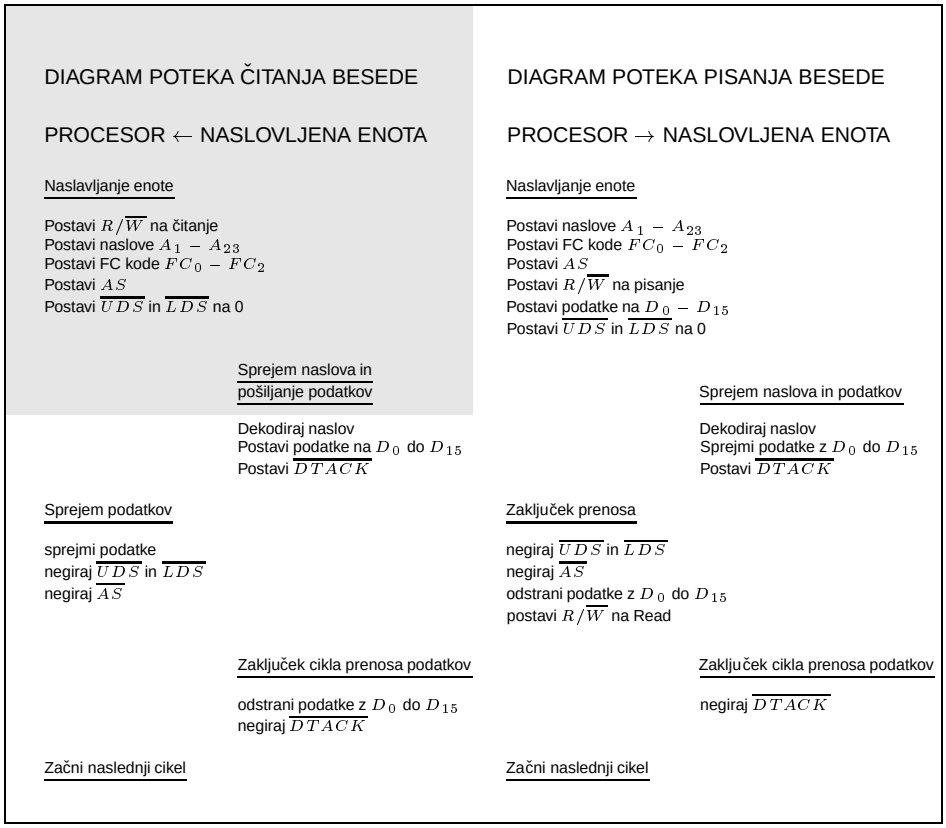


Opomba: v prikazanem primeru je $\overline{DTACK}$ v nasprotju z gornjim principielnim modelom aktiven ob nizkem nivoju.

Potek asinhronega prenosa podatkov je podan na primeru *MC68000*. Na sliki je prikazan protokol čitanja besed iz pomnilnika s časovnim diagramom signalov, v sledeči tabeli pa z opisom posameznih dejanj na strani procesorja in naslovljene enote. V časovnem diagramu so podani zaporedoma trije primeri prenosa podatkov, čitanje in pisanje besede brez čakalnih ciklov ter počasno čitanje z dvema čakalnima cikloma.



Časovni intervali $S_0..S_7$ ustrezajo polperiodam sistemske ure CLK, po kateri deluje mikroprocesor, vendar se ne uporablja za sinhronizacijo prenosa podakov in je pomnilnik pravloma ne vidi. Cikel prenosa podatkov traja najmanj štiri urine periode oziroma osem polperiod. V času S_0 so vsi krmilni signali razen funkcijskih kod FC_x, ki podajajo tip cikla prenosa podatkov, nedefinirani. V polperiodi S_1 se postavijo naslovne linije $R/\overline{W}$, v polperiodi S_2 pa AS (Address Strobe, potrditev veljavnosti naslovov)) in $\overline{UDS}$, $\overline{LDS}$ (Upper in Lower Data Strobe ki podajata dolžino podatka in kateri del podatkovnega vodila bo uporabljen za prenos). S tem preda procesor nadzor nad ciklom naslovljeni enoti. Ta dekodira naslov, in, če se prepozna, vrne podatke, katerih veljavnost potrdi po času za dostop t_{ACC} z $\overline{DTACK}$.
Če procesor prejme $\overline{DTACK}$ vsaj 20ns pred padajočo fronto S_4, bo prečital podatke, ne da bi podaljšal cikel prenosa podatkov. Če pa v tem času signal $\overline{DTACK}$ še ni aktiven, bo procesor *vrival* čakalne cikle, dokler se ne pojavi, kar je prikazano v tretjem delu diagrama na sliki. S tem se cikel prenosa podatkov podaljšuje; v primeru, da se $\overline{DTACK}$ ne bo nikoli zgodil, npr. zaradi okvare ali napačnega naslova, procesor cikla ne bo nikoli zaključil.
Ta situacija predstavlja potencialno nevarnost za izpad sistema, ki je pogojena z asinhronizmom prenosa podatkov in jo moramo rešiti izven procesorja. To običajno storimo z merjenjem časa med AS in $\overline{DTACK}$; če ta čas prekorači neko predvideno vrednost, ki bistveno presega najslabši možni primer, merilno vezje[1] sproži signal, ki procesorju sporoči napako na vodilu ($\overline{BERR}$).
Zaključek cikla je polperioda S_7, ko procesor umakne $\overline{UDS}$, $\overline{LDS}$ in $\overline{AS}$, ter s tem zahteva od periferije tudi umik $\overline{DTACK}$ in podatkov s podatkovnih linij, kar se zgodi v prvi polperiodi naslednjega cikla.



MC68000 ima tudi poseben tip cikla, ki se imenuje *Read-Modify-Write cikel*. V tem tipu cikla se podatki prečitajo iz periferne enote, spremenijo in zapišejo nazaj na isti naslov. Poznamo dva tipa takšnih ciklov:

- pri ukazih za rotiranje in premik operandov, ki ležijo v pomnilniku, se read-modify-write cikel lahko prekine med operacijama čitanja in pisanja;
- posebno obliko tega cikla najdemo pri *Test-And-Set* ukazu (TAS), ki se uporablja za realizacijo semaforjev in podobnih sinhronizacijskih konstruktov, ki jih uporabljamo v multiprogramskih okoljih. Preprečiti želimo, da bi en proces prečital neko zastavico (*flag*), nato bi ga prekinil drugi z višjo prioriteto, prečital isto zastavico, in jo spremenil. Prvi proces bi po zaključku drugega in reaktivaciji delal naprej z že prej prečitano, a sedaj napačno vrednostjo te zastavice. Cikel izgleda kot sekvenca čitalnega in pisalnega cikla, s tem, da se $\overline{AS}$ in naslovni signali med cikloma ne spremenijo, torej je cikel celota in s tem *nedeljiv*.

4.4 Krmilne linije

- signali za nadzor sistema (Clock, Reset, Halt, BusError)

- signali za prenos podatkov (Nasl., podatkovne linije, protokol)

- statusni signali (funkcijske kode)

- signali za dodeljevanje vodila

- prekinitveni signali (proženje prekinitev, potrditev prekinitve)

- napajalne linije (močnejše, pomembna razporeditev modulov)

4.5 Podatki na vodilih

- organizacija: preslikava daljših podatkov na vodila,

- little-big endians,

4.6 Višje funkcije vodil

Poleg osnovne funkcije prenašanja podatkov skrbijo standardni protokoli vodil še za nekatere višje funkcije. Običajno se te zahteve pojavijo, ko lahko imamo na enem vodilu več potencialnih gospodarjev, ki lahko istočasno zahtevajo prenos podatkov. Da ne bi prišlo do kolizije med njimi, je treba z arbitražo reševati te konflikte.

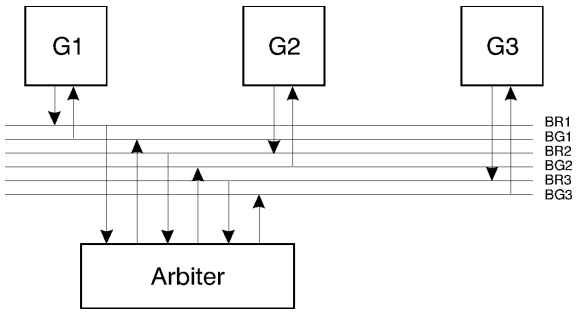
Podoben problem nastopi, ko lahko več naprav istočasno proži prekinitve na enem mikroprocesorju. Ker se vsi ti signali združijo na eno prekinitveno linijo na mikroprocesorju (oziroma na en sistem linij), moramo zagotoviti, da se bo na njegovo poizvedovanje o viru prekinitve odzval le en vir naenkrat.

4.6.1 Dodeljevanje vodil

Algoritmi dodeljevanja *(arbitraža)* morajo zagotoviti, da vsak gospodar, ki to zahteva, po nekem času dobi dostop do vodila. Ti algoritmi se razlikujejo tudi po tem, kdaj bo nek modul sprostil vodilo: ko bo končal svoje delo *(Release When Done, RWD)* ali se ga da prekiniti na zahtevo *(Release On Request, ROR)*.

Obstaja več tipov mehanizmov za arbitražo, med katerimi so najbolj značilni in največkrat uporabljeni zvezdasta arbitraža, marjetična veriga in arbitražno vodilo.

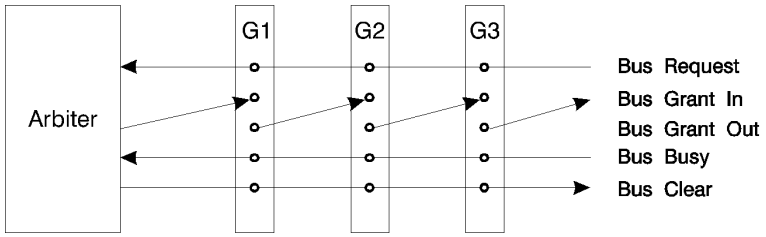
Zvezdasta arbitraža: pri zvezdasti arbitraži ima vsak partner svoji liniji do procesorja za zahtevo po vodilu in odobritev vodila, zaradi česar dodeljevalni algoritem ni vezan na kakšno arhitekturno omejitev; mogoče je realizirati najširši nabor algoritmov dodeljevanja. Ko potrebuje vodilo, modul aktivira linijo Bus Request BR_x; če dobi istočasno več zahtev, se arbiter odloči za eno in temu modulu pošlje odobritev Bus Grant BG_y, ostale pa odloži na kasnejši čas.



Slaba stran tega načina je, da je potrebnih veliko dodatnih linij na vodilu (po dve na vsak modul, ki lahko zahteva prenos) in da obstaja relativno zahteven centralni arbiter, ki predstavlja ranljivo točko v sistemu. V primeru, da ta odpove, sistem namreč ne more več delovati.

Marjetična veriga: za arbitražo na osnovi marjetične verige je potrebnih najmanj linij in tudi najmanj dodatnih vezij. Moduli so vtaknjeni na paralelno vodilo, pri čemer je v najbolj levem (ali desnem) vtičnem mestu poseben modul - arbiter. Pri vodilu je ena linija speljana tako, da ustvari zaporedno povezavo med njimi. Moduli, ki vodila ne potrebujejo, spojijo sponki med $BusGrantIn$ in

$BusGrantOut$, tisti, ki želijo komunicirati, pa jih odprejo.

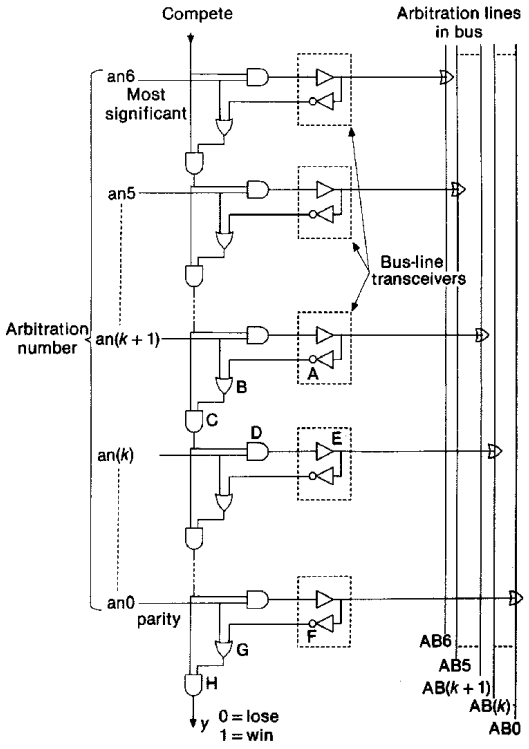


Modul, ki potrebuje vodilo, aktivira linijo $BusRequest$ ter razklene $BusGrantIn$ in $BusGrantOut$; arbiter pošlje $BusGrantOut$, ki potuje po t.i. "marjetični verigi" do prvega modula, ki je zahteval vodilo. Ker je ta razprl sponke, se signal ne širi dalje. Ko vodila več ne potrebuje, spet sklene sponke, zaradi česar lahko dobi $BusGrantIn$ naslednji od čakajočih modulov.

Modul, ki vodilo trenutno uporablja, aktivira linijo $BusBusy$ in jo na koncu sprosti, s čemer pove, da je spet na voljo. Če ga uporablja predolgo, lahko arbiter z $BusClear$ zahteva, da ga sprosti prej. Na ta način lahko realiziramo RWD ali ROR arbitražo.

Slabost marjetične verige je v tem, da lahko realiziramo le prioritetni algoritem dodeljevanja vodila. Fizični položaj modula na vodilu tudi določa njegovo prioriteto pri dostopu do vodila. To pa je običajno težko določiti vnaprej. Obstaja tudi nevarnost, da dodelimo visoko prioriteto modulu, ki bo veliko uporabljal vodilo, zaradi česar moduli za njim ne bodo prišli na vrsto. Nadaljnja slabost je, da morajo biti vsa vtična mesta od levega roba vodila, kjer je arbiter, pa do zadnjega modula zasedena, sicer se marjetična veriga prekine; to pa otežuje hlajenje in smiselno razporeditev modulov glede na njihovo porabo energije.

Arbitražno vodilo Elegantni način arbitraže je izveden s pomočjo posebnega vodila. Tudi pri tem načinu gre za prioritetno dodeljevanje, vendar obstajajo ukrepi, da se njegove slabe strani bistveno omilijo.



Moduli so priključeni na arbitražno vodilo. Vsak od njih ima svojo identifikacijsko številko, ki obenem pomeni tudi njegovo prioriteto in se lahko spreminja, če ugotovimo, da smo jo narobe ocenili; nižje številke pomenijo višjo prioriteto. Vodilo je tipično sestavljeno iz sedmih linij, ki omogočajo identifikacijske kode od 0 do 127. Ko pride do tega, da bi več modulov istočasno želelo prenašati podatke, se sproži postopek tekmovanja za vodilo.

Vsak od konkurenčnih modulov najprej postavi najvišji bit $an6$ svoje identifikacijske kode na najvišjo linijo $AB6$. Če je vrednost tega bita 0, je s tem stanje linije postavil na nizek nivo, če pa je vrednost 1, pusti linijo na visokem. Potem preveri, kakšno je stanje, ko so vsi postavili svoje bite. Če je kateri od modulov postavil 0, je s tem definiral nizek nivo in povedal, da ima višjo prioriteto. Moduli, ki so postavili 1 in nato ugotovili, da je stanje linije na tem nivoju 0, so bili premagani in se umaknejo. Ko je tekmovanje na najvišjem nivoju končano, se nadaljuje na naslednjem ($an5$ in $AB5$) po istih pravilih. Ker imajo moduli različne identifikacijske kode, bo le en ostal po koncu tekmovanja na vseh nivojih; temu se dodeli

vodilo.

4.6.1.1 Prekinitve na vodilih

Podoben problem kot pri dodeljevanju vodila tekmujočim modulom nastopa tudi pri prekinitvah. Na vodilu imamo module, ki lahko prožijo prekinitve (prekinjevalce) in takšne, ki prekinitve strežejo (strežnike prekinitev). Ko prekinjevalec sproži prekinitev v obliki signala na določenih linijah, je to za strežnik znak, da mora ugotoviti, kdo in kaj je zahteval, ter začeti strežbo. Lahko se zgodi, da več prekinjevalcev zahteva strežbo v istem trenutku. Takrat se mora nek arbitracijski mehanizem odločiti, katero zahtevo bo prvo postregel. Ti mehanizmi so podobni zgoraj opisanim. Pri komercialnih vodilih najpogosteje nahajamo mehanizem z marjetično verigo (npr. VME).

4.6.2 Skupine signalov na vodilih

Signale na vodilih lahko razdelimo na nekaj skupin. Nekatere od teh, na primer *podatkovni, naslovni signali ter signali za krmiljenje poteka podatkov* so nam znane že iz poglavij o prenosu podatkov. V gornjem podpoglavju smo omenjali *signale za arbitražo* in *prekinitvene signale*. Poleg teh imamo še nekatere *sistemske signale*, kot so urini in sinhronizacijski signali, reset, halt ipd., ter napajalne linije, preko katerih pripeljemo na module različne napetosti in maso. Posebej omenjamo še *signale za odkrivanje in popravljanje napak* (npr. pariteta) in manj običajne *pozicijske signale*, ki služijo za to, da lahko ugotovimo konfiguracijo sistema in položaj posameznih modulov na vodilu.

4.6.3 Funkcijski moduli in opcije vodil

Standardi na področju vodil so razmeroma široko zasnovani in pokrivajo cel spekter aplikacij. V popolni izvedbi zato zahtevajo razmeroma veliko linij in sklopov strojne opreme, ki podpirajo določene možnosti. Pogosto pa se zgodi, da nekaterih funkcij, ki so sicer definirane s standardom, ne potrebujemo in njihova izvedba samo zaradi skladnosti s standardom ne bi bila smiselna. V ta namen so posamezne funkcije, ki jih podpira vodilo, razdeljene v funkcijske module, standard pa

dovoljuje, da se odločimo, katere bodo v dejanski aplikaciji v resnici uporabljene in katere ne.

Opcije vodil ne nanašajo na primer na

- širino podatkovnega in naslovnega vodila,
- izvedbo z enim ali več gospodarji,
- strategijo in podrobnosti arbitraže,
- enim ali več prioritetnimi nivoji za dodeljevanje vodila in prekinitve,
- varnostne mehanizme,
- pomožna serijska vodila, itd.

Ti funkcijski moduli so običajno izvedeni v obliki ASIC (namenskimi integriranimi vezji - Application Specific Integrated Circuits) v različnih tehnologijah. Vgradimo jih v vtične module, ki jih vključimo v vodilo, nekatere pa v primeru kompleksnejših vodil združimo na posebnem vtičnem modulu, namenjenem krmiljenju vodila, na katerem so npr. sistemska ura, arbiter, krmilnik za pomožna vodila, nadzor napajalnika, modul za nadzor prenosa podatkov ter ugotavljanje in morda celo popravljanje napak, itd.

4.7 Primeri vodil

Delitev na notranja in zunanja vodila:

– notranja: neposredno povezujejo procesor z drugimi osnovnimi enotami (pomnilnik, periferni vmesniki); hitra, paralelna, preprostejši protokoli

– zunanja: povezava mikroračunalnika z zunanjimi enotami (zunanja periferija, zunanji diski, telekomunikacije..)

4.7.1 Notranja vodila

4.7.1.1 Vodilo IEEE 1014 - VME

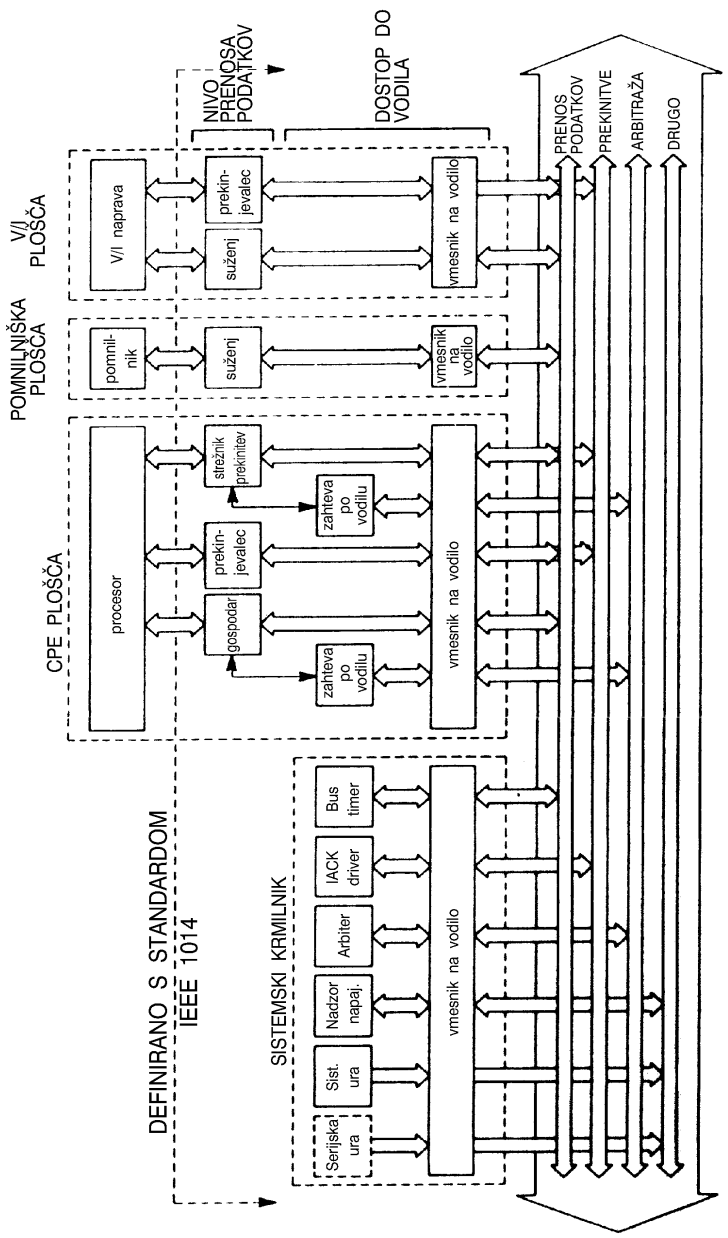
Kot primer bomo na kratko predstavili zelo razširjeno vodilo VME, ki je bilo leta 1985 privzeto kot standard IEEE 1014. Izhaja iz Motorolinega vodila VersaBus iz konca 70 let, ki so ga prilagodili tudi drugim uporabnikom.

Vodilo je sestavljeno iz štirih skupin linij. Prva skupina je namenjena *prenosu podatkov*. Prenašajo se lahko podatki širine 8, 16 ali 32 bitov s 16, 24 ali 32 bitnimi naslovi po asinhronem protokolu. Druga skupina predstavlja *arbitracijsko vodilo*, ki omogoča dodeljevanje na osnovi kombinacije prioritet in marjetične verige: na voljo so 4 hierarhični nivoji, na katerih so zgrajene verige. Mogoča sta načina sproščanja vodila na zahtevo (ROR) in po zaključku (RWD). Na *prekinitvenem vodilu* je sedem prekinitvenih linij in marjetična veriga za izvedbo prekinitvenega prevzemnega cikla. Ostale linije tvorijo *pomožno vodilo*. To so napajanja, inicializacijski signali in diagnostika.

Sistemska shema je podana na sliki. Prikazani so funkcijski moduli, ki so definirani v standardu VME 1014. Nekateri od njih se (lahko) nahajajo na posebni plošči, ki se imenuje *sistemski krmilnik vodila* (System Bus Controller), in predstavljajo osnovne sistemske funkcije:

- serijska in sistemska ura: modula generirata ustrezne sinhronizacijske signale;
- nadzor napajalnika: ob izpadu omrežne napetosti generira signal, ki ga sistem poskusi postreči, dokler je še energija v napajalniku;
- arbiter: dodeljevalec vodila po principu marjetične verige;
- IACK driver: gonilnik marjetične verige za cikel potrjevanja prekinitve;
- bus timer: nadzoruje prenos podatkov in rešuje situacije, ko v predvidenem času ni odziva na začetek cikla prenosa podatkov.

Module, ki sodelujejo v prenosu podatkov, lahko razvrstimo na dva nivoja: nivo prenosa podatkov (višji) in nivo dostopa do vodila (nižji). Dostop do vodila omogočata naslednja modula:


NIVO PRENOSA PODATKOV
DOSTOP DO VODILA
V/I PLOŠČA
V/I naprava
prekin-jevalec
sužnenj
vmesnik na vodilo
POMNILNIŠKA PLOŠČA
pomnil-nik
sužnenj
vmesnik na vodilo
CPE PLOŠČA
procesor
strežnik prekinitev
prekin-jevalec
gospodar
zahteva po vodilu
zahteva po vodilu
vmesnik na vodilo
DEFINIRANO S STANDARDOM IEEE 1014
SISTEMSKI KRMILNIK
Bus timer
IACK driver
Arbiter
Nadzor napaj.
Sist. ura
Serijska ura
vmesnik na vodilo
PRENOS PODATKOV
PREKINITVE
ARBITRAŽA
DRUGO

- vmesniki na vodila: opravljajo preprosto fizično funkcijo priključitve na linije vodila;

- modul za zahtevo po vodilu; z njim iniciator prenosa (gospodar) zahteva od arbitra dodelitev dostopa do vodila oziroma sproži arbitražo.

Na nivoju prenosa podatkov so moduli:

- gospodar: nadrejeni aktivni modul, ki zahteva prenos podatkov. Ko dobi vodilo, ga tudi izvede;

- suženj: sodeluje v prenosu podatkov;

- prekinjevalec: lahko pošlje zahtevo po prekinitvi;

- strežnik prekinitev: jo mora postreči. Ker mora izvesti prekinitveni prevzemni cikel, potrebuje dostop do vodila. Prav tako potrebuje strežbo gonilnika IACK linije, ki odloči, katero prekinitev bo treba najprej postreči.

Ker je za manj zahtevne aplikacije lahko takšno vodilo prekompleksno, obstajajo standardizirane *opcije*, ki omejujejo osnovni standard. Tako je lahko podatkovno vodilo široko tudi le 8 ali 16 bitov, oziroma naslovno 16 ali 24 bitov. Pri arbitraži se lahko odpovemo sproščanju vodila na zahtevo (ROR), lahko delamo le s fiksnimi prioritetami ali pa z marjetično verigo le na enem nivoju. Lahko se tudi odpovemo nekaterim prekinitvenim nivojem, itd.

4.7.1.2 PCI

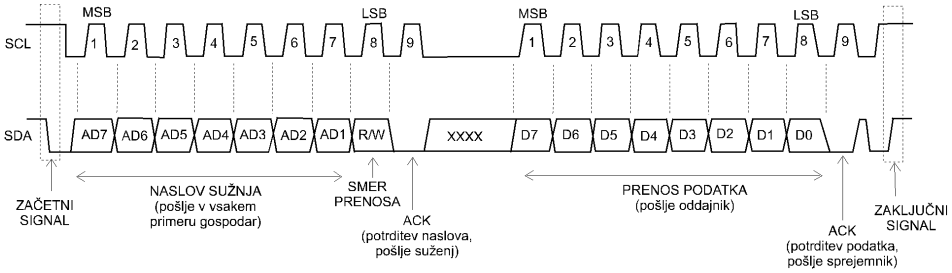
4.7.1.3 ISA

4.7.1.4 itd..

4.7.2 Zunanja vodila

4.7.2.1 Serijsko vodilo I^{2}C (M-Bus)

Pri serijskem vodilu se podatki prenašajo časovno multipleksirano po eni sami podatkovni liniji. Primer takšnega vodila je npr. Philipsovo I^2C vodilo, ki ga je Motorola kasneje prevzela kot svoj M-bus. Sestavljata ga liniji za podatke in sinhronizacijo (SDA – Serial Data in SCL – Serial Clock). Na to vodilo so paralelno priključene različne naprave, ki paroma komunicirajo med seboj. Obstajata dva tipa modulov, aktivni (gospodar - master), ki lahko sprožijo prenos, in pasivni (suženj - slave), ki lahko samo odgovarjajo na zahteve. Standardna serijska komunikacija je sestavljena iz štirih delov: začetnega signala, naslova naslovljene suženjske enote, prenosa podatkov in končnega signala.



Ko je vodilo prosto (SDA in SCL sta obe na visokem nivoju), lahko gospodar začne komunikacijo tako, da sproži t.i. *začetni signal*: ob visokem SCL postavi SDA na nizek nivo, s čemer začne protokol in vzbudi vse sužnje. Takoj za to sekvenco pošlje sedem-bitni naslov sužnja, s katerim želi komunicirati, in smerni bit. Vsak bit informacije, oddan na SDA, je veljaven in mora biti stabilen ob visokem nivoju SCL; podatek se lahko spremeni le, ko je SCL na nizkem nivoju. Suženj, katerega naslov se ujema z oddanim, ob devetem impulzu SCL potrdi svojo prisotnost in pozornost s tem, da postavi linijo SDA na nizek nivo. Ko je zveza

uspešno vzpostavljena, se začne prenos podatkov v zahtevani smeri: ob SCL, ki ga generira gospodar, pošiljata podatke na SDA gospodar ali suženj. Vsak oddan paket osmih bitov mora potrditi sprejemnik z ACK bitom ob devetem impulzu SCL. Če izostane potrditev s strani sužnja, gospodar pošlje zaključni signal in s tem prekine oddajo, ali pa začetni signal za ponovni prenos. Če pa prejema ne potrdi gospodar, mora suženj sprostiti SDA linijo in prekiniti protokol.

Urin signal v vsakem primeru generira na SCL gospodar, ne glede na smer prenosa. Po osnovnem standardu je največja hitrost prenosa omejena na 100 kbit/s, njegova razširitev pa dovoljuje tudi višje hitrosti . Suženj lahko zadrži oddajanje naslednjega paketa po potrditvi, če potrebuje čas za obdelavo sprejetega podatka. To naredi tako, da zadrži SCL po devetem impulzu na nizkem nivoju. Ko gospodar želi nadaljevati naslednji paket s prvim impulzom SCL, ugotovi, da ja le-ta na nizkem nivoju in počaka, da se sprosti.

Gospodar konča prenos z zaključnim signalom, ki je določen s prehodom SDA z nizkega na visoki, medtem ko je SCL na logični visokem nivoju; s tem se liniji sprostita.

Serijsko vodilo I^2C oz. M-Bus omogoča tudi priključitev več gospodarjev. Preden začne uporabljati vodilo, gospodar preveri, ali je prosto; to ugotovi tako, da preveri, ali sta obe liniji na visokem nivoju. Če se zgodi, da dva ali več gospodarjev istočasno ugotovijo, da je vodilo prosto, pride do trka; tega je treba razrešiti z arbitražo.

Arbitraža poteka vzporedno s prenosom podatkov. Najprej gospodarji, ki so v konfliktu, sinhronizirajo svojo uro: tisti, katerega ura bi morala prej preiti z nizkega na višji nivo, pa jo nekdo drug še vedno zadržuje na nizkem, počaka tako, da postavi svoj izhod SCL na stanje visoke upornosti; šele ko ura zadnjega gospodarja preide v visoko stanje, začnejo spet vsi meriti čas in generirati urine impulze.

Naslovniki imajo svoje identifikacijske naslove; tisti z nižjo vrednostjo ima višjo prioriteto. Vsak gospodar opazuje vodilo medtem, ko nanj postavlja bite naslova partnerja, od najpomembnejšega bita nazaj.

Podatkovna linija je tipično na visokem nivoju; če je vrednost bita, ki ga želi gospodar prenesti, enaka 1, jo v tistem urinem ciklu pusti pri miru, če pa je 0, jo sklene na maso. Če torej dva gospodarja postavita različna bita istočasno na

linijo, bo na njej prevladal tisti, ki je nanjo vpisal 0 (oz. jo sklenil proti masi).

Če gospodar ugotovi, da je v primeru, ko je na podatkovno linijo postavil enico, na njej našel ničlo, preneha z oddajanjem in se umakne. S tem prepusti vodilo tistemu (ali tistim), ki pošilja(jo) sporočilo partnerju z višjo prioriteto, sam pa poskusi pošiljati ponovno, ko se vodilo spet sprosti. Če dva ali več gospodarjev pošiljajo sporočilo istemu sužnju (isto naslovno polje v sporočilu), se arbitraža nadaljuje še v podatek.

Postopek je podoben arbitraži z arbitražnim vodilom, ki je podrobneje opisan v poglavju 4.6.1.

4.7.2.2 SCSI

4.7.2.3 CAN

Poglavje 5

Pomnilnik

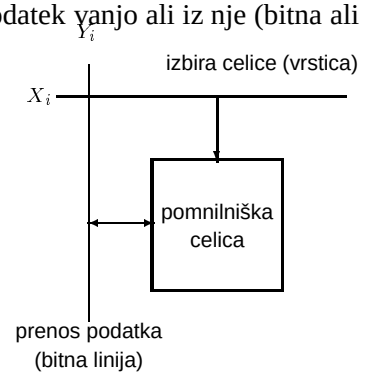
Pomnilnik je ena od temeljnih enot računalnikov; v njem hranimo podatke in programe. V klasični ali von Neumannovi arhitekturi se programi nahajajo fizično v istem pomnilniškem prostoru kot podatki. Posledica tega je, da moramo vanj posegati zelo pogosto, za dostop do vsakega ukaza in zahtevanih operandov; zaradi tega postane prenos podatkov po vodilu iz pomnilnika oz. vanj ozko grlo. Ločitev pomnilnikov na podatkovne in programske in ločena izvedba prenosnih poti do njih –vodil– (npr. Harwardska arhitektura) lahko to ozko grlo omili.

Bistveni del pomnilnika je *pomnilni element*. To je element, ki je sposoben za določen čas ohraniti neko stanje (informacijo). Primeri takšnih elementov so shrambe energije (npr. kondenzatorji in tuljave, pri katerih se naboj oz. magnetni pretok ohrani tudi po odklopu vira energije), magnetni mediji (trakovi, diski), elementi, ki sami vzdržujejo stanje (npr. bistabilni multivibratorji ali flip-flopi), elementi, ki jim informacijo vgradimo pri izdelavi (npr. z masko pri ROM pomnilnikih), ipd. Nekateri od teh elementov vzdržujejo informacijo trajno, drugi dokler so napajani, tretji zelo kratek čas, zaradi česar je treba njihovo stanje neprestano obnavljati.

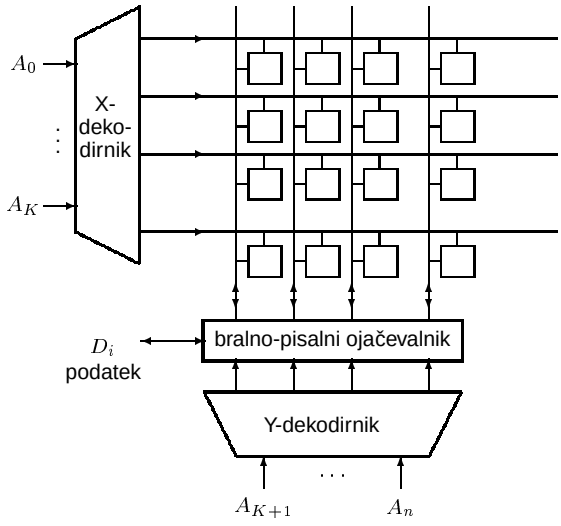
V nadaljevanju se bomo omejili na polprevodniške pomnilnike, ki predstavljajo bistvene elemente mikroračunalnikov. Delimo jih na bralno-pisalne (read/write) in samo bralne (read-only) pomnilnike. Pri bralno-pisalnih je vsaka celica dostopna za pisanje in čitanje, bralne pa lahko v normalnem delovanju le čitamo, vsebino jim vpišemo v posebnem postopku ob snovanju sistema - programiranju.

5.1 Zgradba pomnilnikov

Pomnilnik je sestavljen iz *pomnilniških celic*, ki hranijo po en bit informacije. Vsaka celica ima dva priključka; z enim celico izberemo ali naslovimo (X_i), z drugim pa prenesemo podatek vanjo ali iz nje (bitna ali podatkovna linija Y_i):



Celice so v pomnilniku različno organizirane. Najpogostejša je matrična organizacija, pri kateri so razporejene v vrstice in stolpce. Matrika je običajno kvadratna, lahko pa je tudi pravokotna. Pri tem se izbirajo celice po vrsticah, podatki pa prenašajo po stolpcih.



Naslov celice pomnilnika je sestavljen iz naslova vrstice in naslova stolpca, ki sta pri kvadratnih matrikah enako dolga. Naslov vrstice $A_{0..k}$ gre v X-dekodirnik, ki glede nanj aktivira enega od izbirnih signalov in z njim izbere celo vrstico celic pomnilnika. Vsebina teh celic se po bitnih linijah prenese preko ojačevalnkiov v bralno-pisalni register. Drugi del naslova $A_{(k+1)..n}$ se dekodira v Y-dekodirniku. Če gre za čitanje podatkov, se podatek iz naslovljenega mesta bralno-pisalnega registra (ki pomeni stolpec v matriki) prenese na izhod D_i. V primeru pisanja se po prenosu izbrane vrstice v bralno-pisalni register podatek z D_i vpiše na izbrano mesto, od koder se cela vrstica prenese v nazaj pomnilne celice. Biti iz stolpcev, ki niso bili naslovljeni, se ne spremenijo.

Prikazana matrična organizacija predstavlja tako imenovano “bitno ravnino”, ki je primerna za shrambo enobitnih podatkov. Kadar želimo organizirati pomnilnik po večbitnih besedah, združimo več bitnih ravnin. Istoležne celice v različnih ravninah naslovimo z istimi naslovi; podatek na linijah $D_{1..j}$ predstavlja naslovljeno besedo, pri čemer pride vsak bit podatka z ene bitne ravnine.

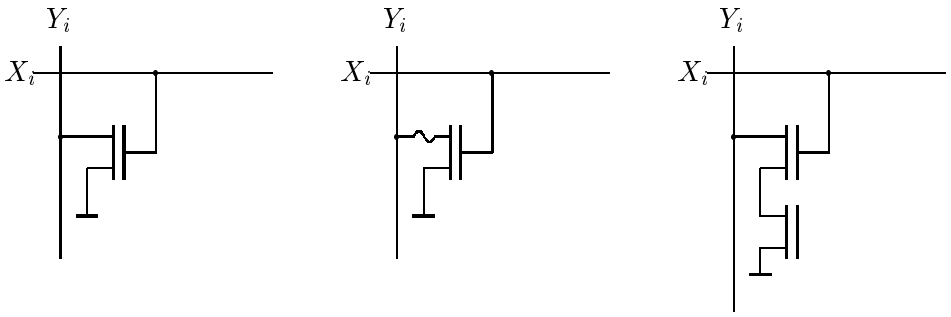
Včasih je ugodno izkoristiti, da pri matrični organizaciji vedno posegamo do celotnih vrstic. Kadar na primer prenašamo iz pomnilnika cele bloke podatkov, najprej izberemo vrstico. Ta se prenese v bralno-pisalni register. Sedaj prenesemo iz njega ne le podatek, ki pripada naslovljenemu stolpcu, temveč zaporedoma vse podatke. Na ta način lahko prenesemo veliko podatkov ob le enem naslavljanju vrstice, kar bistveno poveča hitrost dostopa. Podobno lahko pri vpisovanju bloka podatkov vpišemo podatke v vse bite bralno-pisalnega registra in ga z eno samo operacijo prenesemo v celice v izbrani vrstici. Ta, tako imenovani način dostopa po straneh (*page mode operation*), zahteva, da je velikost blokov podatkov prilagojena velikosti strani.

Poleg matrične poznamo še linearno organizacijo pomnilnika. Pri tej imamo le eno vrstico, ki je vedno dostopna v bralno-pisalnih ojačevalnikih in je torej ni potrebno šele izbrati in prenesti tja. Takšna organizacija zagotavlja hiter dostop do podatkov, vendar je zaradi kompleksnosti Y dekodirnika primerna za manjše kapacitete pomnilnikov.

5.2 Vrste pomnilniških celic

5.2.1 Bralni pomnilniki

Bralni pomnilniki so razmeroma preprosti; njihove celice sestavlja po ena dioda ali tranzistor. V **ROM** (Read-Only Memory), najpreprostejšega med njimi, vpišejo informacijo že ob njegovem snovanju (t.j. izdelavi maske), tako, da tam, kjer naj bo informacija "0", tranzistor ali diodo (na sliki je prikazan primer s tranzistorjem) vgradijo, kjer naj bo "1", pa ne. Z izbiro vrstice (linija X_i povzroči, da tranzistor prevaja) tranzistor (če obstaja) napetost na liniji y_j postavi na nizek nivo. Proizvodnja pomnilnika z uporabo ROMov je poceni, saj jih ni treba programirati; tudi izdelava samih čipov je poceni, draga pa je izdelava maske, saj jo je potrebno izdelati za vsako vsebino pomnilnika posebej. Zato se ROM uporabi tam, kjer je predvidena velika serija in se cena izdelave maske porazdeli na veliko število kosov.

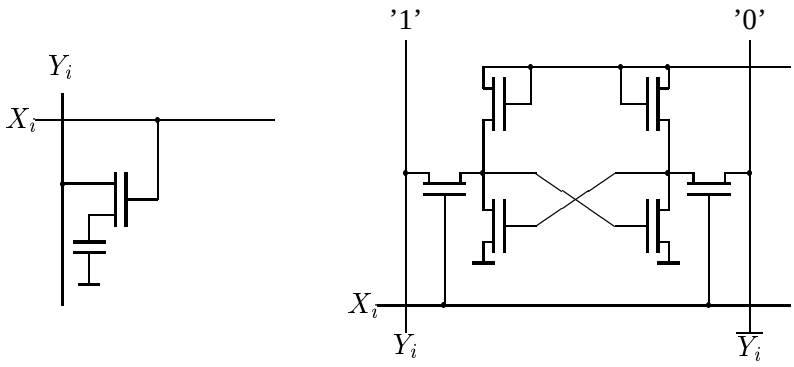


Za manjše serije je ugodno uporabiti **PROM** (Programmable Read-Only Memory). To so univerzalni pomnilniki, ki imajo ob proizvodnji na vseh mestih predvidene tranzistorje; ti imajo vgrajeno "varovalko", šibko mesto, ki ga lahko z višjo napetostjo prežgemo in s tem izključimo tranzistor. Programiranje PROMov je enkratno dejanje: ko je varovalka prežgana, je ni mogoče ponovno vzpostaviti. Proizvodnja PROMov je zaradi univerzalnosti velikoserijska, ne zahteva izdelave posebne maske in je zato poceni, vendar je treba ob uporabi v aplikaciji vsakčip posebej programirati, kar podraži proizvodnjo končnega izdelka.

Za laboratorijsko in maloserijsko uporabo so namenjeni **EPROMi** (Erasable Programmable Read-Only Memory), ki jih lahko sprogramiramo, po potrebi pa tudi zbrišemo njihovo vsebino z ultravijolično svetlobo. Ti imajo v vsaki celici vgra-

jen poseben element (Floating-gate Avalanche-Injection MOS ali Stacked Gate Memory Cell), ki ga z višjo programirno napetostjo postavimo v prevodno stanje. V tem stanju ostanejo tudi po izklopu napajanja, v neprevodno pa jih spravimo z obsevanjem z ultravijolično svetlobo. EPROMi so relativno dragi, samo uporabo za realizacijo pomnilnikov pa podraži tudi dejstvo, da jih je treba programirati. Primerni so za razvoj, prototipe in zelo male serije.

Kadar moramo zagotoviti, da je informacija ob vklopu napajanja že prisotna, obenem pa mora biti mogoče vsebino celic spreminjati med samim delovanjem, uporabimo **EEPROM** (Electrically Erasable Programmable Read-Only Memory). Teh ni potrebno brisati s svetlobo, temveč lahko to dosežemo z električnimi signali; vpis posameznih celic je mogoč v samem vezju in brez posebnega programatorja. EEPROM pa ne more nadomestiti pravih bralno-pisalnih pomnilnikov, saj je pisanje približno 20.000-krat počasnejše od čitanja, pa tudi število vpisov v celico je omejeno (življenjska doba je le okoli 10.000 vpisov).



5.2.2 Bralno–pisalni pomnilniki

Bralno-pisalne pomnilnike (RAM – Random Access Memory, pomnilniki s poljubnim dostopom) delimo na *statične in dinamične*.

Statični RAM ponilnik ohrani vsebino, ki jo vpišemo vanj, dokler je priključeno napajanje. Njegov pomnilni element je bistabilni multivibrator (flip-flop); pomnilniška celica je običajno priključena na dve podatkovni (bitni) liniji, ki nosita invertirano stanje: ko je podatek "0", je aktivna $\overline{y_i}$, kadar je "1", pa y_i. Z izbirno linijo x_j odpremo vrata na ti liniji in s tem prečitamo oz. vpišemo nov podatek. Delovanje statičnega bralno-pisalnega pomnilnika je razmeroma hitro, vendar je

za eno celico potrebnih šest tranzistorjev. Zaradi tega je število celic na površino silicijevega čipa razmeroma majhno, cena pomnilniške kapacitete pa velika. Ker flip-flopi ohranijo informacijo, dokler so napajani, njihove vsebine ni potrebno obnavljati, kar poenostavi uporabo v primerjavi z dinamičnimi pomnilniki.

Pomnilni element v celici **dinamičnega RAM pomnilnika** je parazitna kapacitivnost tranzistorja, ki je na sliki prikazana kot navidezni kondenzator, katerega vrednost je okoli 0,1 pF. Ob vpisu se tranzistor odpre z linijo x_i in kapacitivnost se napolni na napetost, kot je na liniji y_i. Ta majhen naboj se zaradi neidealnih notranjih upornosti začne prazniti; koristna informacija se tipično ohrani le nekaj milisekund. Zato je treba v tem času podatek iz vsake celice dinamičnega pomnilnika prečitati in ponovno vpisati – *osvežiti*.

Čitanje podatka iz dinamičnega RAMa se zaradi njegove narave nekoliko razlikuje od čitanja iz drugih polprevodniških pomnilnikov. Ko vrstični dekodirnik izbere vrstico, se naboj, ki vsebuje informacijo, iz vseh pomnilniških celic prenese preko vertikalnih podatkovnih linij in čutilnih ojačevalnikov v bralno-pisalni register.

Ob vsakem čitanju se kapacitivnosti vseh celic v vrstici izpraznijo - pravimo, da je čitanje destruktivno - in jih je torej potrebno v vsakem primeru takoj vpisati nazaj, saj jih želimo ohraniti. Ta procedura pa ima tudi dober stranski učinek: ne samo ob vpisu, ampak tudi ob čitanju se vsebine vseh celic v vrstici osvežijo.

Celico dinamičnega pomnilnika sestavlja en sam tranzistor, zato je gostota pomnilniških celic na enoto površine čipa razmeroma velika v primerjavi s statičnim pomnilnikom, cena pa ustrezno nižja. Njegova slaba stran je, da je treba skrbeti za osveževanje.

Vsebina dinamičnih RAM pomnilnikov se lahko osvežuje na več načinov. Najpreprostejši način je programsko osveževanje: če program, ki ga izvaja mikroprocesor, deluje v zanki, ki se pogosto ponavlja, je s tem lahko že zagotovljeno, da bodo vse celice v pomnilniku vsaj prečitane in s tem osvežene v določenem časovnem obdobju.

Kadar to ni mogoče, uporabljamo druge tehnike osveževanja. Pri teh moramo z v RAM vgrajenimi ali pa od zunaj dodanimi števci v času, ko pomnilnik ni naslovljen, generirati naslove, ki jih ob potrditvi z RAS pošiljamo na vrstični dekodirnik. Dekodirnik izbere vrstico, vsebine celic se prenesejo v bralno-pisalni

Pravzaprav je delovanje dinamičnih RAM pomnilnikov precej bolj zahtevno zaradi zelo majhnih nabojev, ki jih hranijo parazitne kapacitivnosti v celicah. Signali, ki jih povzročijo, ko stečejo na podatkovne linije, so zelo majhni v primerjavi z motilnimi signali, ki prihajajo iz okolice preko napajanja, po zraku, pa tudi kot posledica delovanja same elektronike v čipu. Signal, ki je nastal kot posledica prenosa naboja iz pomnilniške celice preko podatkovne linije, bi bilo nemogoče neposredno prepoznati izmed ostalih motečih signalov.
Zato se uporablja tehnika, ki temelji na primerjavi napetosti dveh linij: ena je prava podatkovna linija, na katero so priključene pomnilniške celice, druga pa je slepa linija. Ta poteka blizu podatkovnih, zato se na njej pojavijo zelo podobne motnje kot na ostalih. Na njo je priključena slepa celica s podobno strukturo kot jo imajo pomnilniške, le da je njena kapacitivnost za polovico manjša in je vedno napolnjena.
Ob aktivaciji vrstice se sedaj naboji iz pomnilniških celic po koristnih in po slepi liniji prenesejo v čutilne ojačevalnike. Tam se vse podatkovne primerjajo s stanjem slednje: če se je po neki liniji prenesel naboj, bo njena napetost v tistem trenutku višja od tiste na slepi liniji (ki je imela polovico manjši naboj), če pa naboja v pomnilniški celici ni bilo, bo nižja. Čutilni komparatorji izločijo to razliko. Vsi ostali motilni signali, ki so na vseh linijah približno enaki, pa se med primerjavo odštevajo in s tem izničijo.

ojačevalnik in avtomatsko nazaj; nato se operacija konča. Pri tem moramo prešteti vse naslove vrstic v zahtevanem času.

Zaradi velike kapacitete pomnilnika bi moral imeti dinamični RAM veliko število priključkov pri razmeroma majhnem čipu. Da bi se temu izognili, je naslovno vodilo, ki je že po naravi razdeljeno na naslov vrstice in naslov stolpca, praviloma multipleksirano: najprej je na vodilu polovične širine naslov vrstice, ko je vsebina njenih celic že v bralno-pisalnem ojačevalniku, pa naslov stolpca. Kateri del naslova je na vodilu, se pove s signaloma $\overline{RAS}$ (Row-Address-Strobe) in $\overline{CAS}$ (Column-Address-Strobe).
Drugi ukrep za zmanjšanje števila priključkov na čipu je, da so klasični dinamični RAM pomnilniki običajno enobitno organizirani, kar pomeni, da imajo en sam podatkovni priključek. Za besedo poljubne širine moramo torej uporabiti ustrezno število pomnilniških čipov, katerih naslove vežemo skupaj.

5.3 Posebne izvedbe pomnilnikov

Izboljšanje zanesljivosti podatkov v pomnilniku: kadar pričakujemo, da bi lahko zaradi posebnih razmer, v katerih deluje računalnik, prišlo do nezaželene spremembe vsebine pomnilnika in s tem do napake, lahko uporabimo posebne pomnilnike, ki vsebujejo poleg podatkov še redundantno informacijo za ugota-

vljanje napak. Takšni pomnilniki imajo za en bit daljšo besedo kot podatek (npr. 9 bitov za bajt).

Ob vpisu posebno vezje izračuna in v tem redundantnem ("paritetnem") bitu shrani informacijo o parnosti podatka: če je v dejanskem podatku neparno število enic, bo njegova vrednost 1, sicer 0. Pri čitanju se prečita celotna beseda in preveri število enic (skupaj s paritetnim bitom) v njej: to število mora biti vedno parno. Če ni, vezje signalizira napako.

Bralno-pisalni pomnilniki, ki ohranijo stanje ob izklopu napetosti: včasih potrebujemo bralno-pisalni pomnilnik, ki pa si bo zapomnil vsebino ob izklopu napajanja in bo po ponovnem vklopu nadaljeval njo, torej na mestu, kjer je ostal. To lahko zagotovimo na dva načina:

- posebni statični RAM pomnilniki imajo dva režima delovanja: normalno delovanje in stanje pripravljenosti (stand-by mode). V prvem se obnašajo kot običajni pomnilniki, ko pa glavno napajanje ugasne, preidejo v drugi način. V njem so s posebno dodano baterijo napajane le pomnilniške celice; ker so izvedene v varčni CMOS tehnologiji, je za to potrebno zelo malo energije in se vsebina z drobno baterijo ohrani zelo dolgo. Ker so vhodno/izhodni vmesniki brez napajanja in torej izklopljeni, tudi morebitni signali na vhodih ne spreminjajo vsebine pomnilnika.

  Ker so baterije še vedno tehnološko nepopolne in obstaja nevarnost, da stečejo, namesto njih včasih uporabljajo posebne kondenzatorje, ki prav tako zelo dolgo (tipično več mesecev) ohranijo vsebino pomnilnika.

- obstajajo posebni NOVRAM (Non-Volatile RAM) pomnilniki. V teh pomnilnikih sta združena EEPROM in RAM, katerih naslovni področji se prekrivata. V normalnem načinu delovanja je aktiven RAM. Ob aktivnem signalu na priključkih $\overline{STORE}$ oz. $\overline{RECALL}$ pa se vsi podatki v paketu prenesejo iz RAMa v EEPROM oziroma obratno.

  Ti signali se lahko aktivirajo ob poljubni trenutkih, pri čemer zahteve po posegih v RAM med prenosom podatkov v in iz EEPROMa niso dovoljene. Tudi če pride zahteva za $\overline{STORE}$ ali $\overline{RECALL}$ med operacijo čitanja ali pisanja podatkov v RAM, se slednje prekinejo. Absolutna prioriteta teh operacij je pogojena s tem, da se običajno prožijo ob resnih dogodkih v

sistemu, kot je npr. izklop oz. izpad napajanja ali nerešljive sistemske napake.

Prenos vsebine med RAMom in EEPROMom je mogoče avtomatizirati. S posebnim analognim elektronskim vezjem se nadzoruje stanje napajalne napetosti: ko to naraste iz nizke v sprejemljivo, primerjalno vezje generira signal $\overline{RECALL}$; obratno, ko napajalna napetost pade pod predpisan nivo, se sproži $\overline{STORE}$. Ker napajalna napetost običajno ne upade trenutno na 0 in ker kapaciteta NOVRAM pomnilnikov običajno ni velika (nekaj deset do sto kB), je dovolj okoli 10ms časa, v katerem mora biti napetost zadostna, da se lahko shrani ves pomnilnik.

5.4 Naslovni dekodirniki

V gornjih poglavjih je bilo prikazano, kako delujejo pomnilniški elementi, v tem pa se bomo posvetili gradnji pomnilniških modulov, zgrajenih iz njih.

Pomnilniški modul je vezje na plošščici, ki vsebuje več pomnilniških čipov; ti tvorijo pomnilnik z ustreznim številom pomnilniških besed določene dožine. V računalnik lahko praviloma vstavimo več takšnih modulov, s čemer zagotovimo potrebno količino pomnilniškega prostora.

Vsak čip, ki nastopa v modulu, ima določeno količino besed, katerih vsaka ima svoj naslov. Ta naslov je notranji naslov v čipu, njegova dolžina pa je odvisna od njegove kapacitete.

Primer: hipotetični pomnilniški čip kapacitete 2K bytov ima naslovno področje od $000 (%000 0000 0000) do $7FF (%111 1111 1111), kar je 2048_{10} naslovov. Za notranje vodilo torej potrebujemo enajst bitov.

Naslovno vodilo ima praviloma večjo širino, kot je notranji naslov. S pomočjo ostalih bitov določamo, kateri izmed čipov se naj odzove: z naslovnim dekodirnikom generiramo signale, ki izberejo ustrezni čip preko njegovih izbirnih vhodov (običajno aktivnih na nizkem nivoju in imenovanih chip select - $\overline{CS}$, chip enable - $\overline{CE}$ ali podobno).

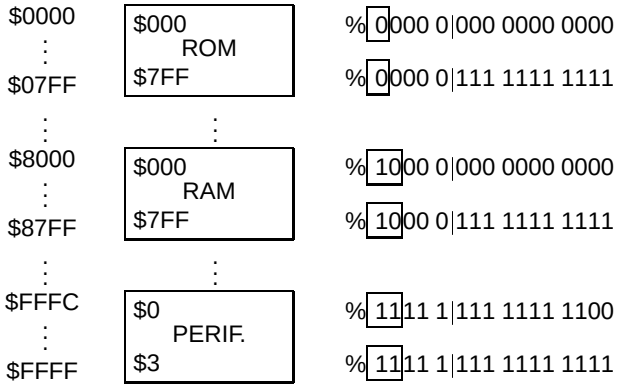
Primer: koliko čipov iz gornjega primera lahko priključimo na šestnajstbitno vodilo?

$$2^{16}/2^{11} = 2^5 = 32$$

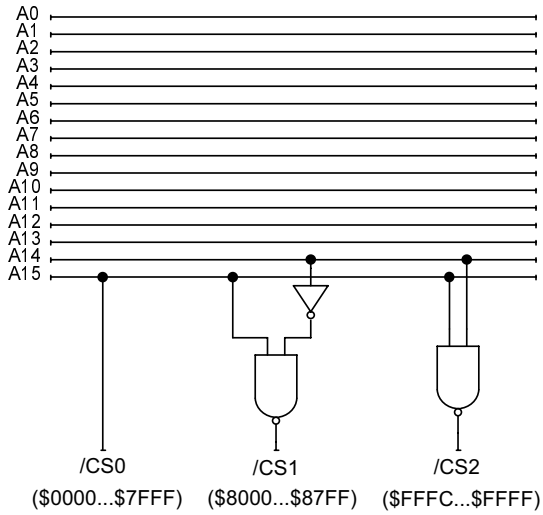
To lahko vidimo tudi iz širine naslovov: če upoštevamo, da notranji naslov zateva 11 od 16 linij vodila, s preostalimi 5 linijami lahko generiramo 32 kombinacij oziroma naslovov čipov.

Podobno kot pomnilnike vključujemo v naslovna področja tudi periferne vmesnike, katerih registri se torej pojavljajo kot posamezne pomnilniške lokacije. Registrov je bistveno manj kot pomnilniških celic, zato so njihova notranja naslovna področja zelo kratka, tipično nekaj besed.

Primer: v računalniku imamo ROM in RAM pomnilnika kapacitete po 2K in periferni vmesnik s štirimi registri na spodnjih naslovih. Kako bo izgledal najpreprostejši naslovni dekodirnik zanje?



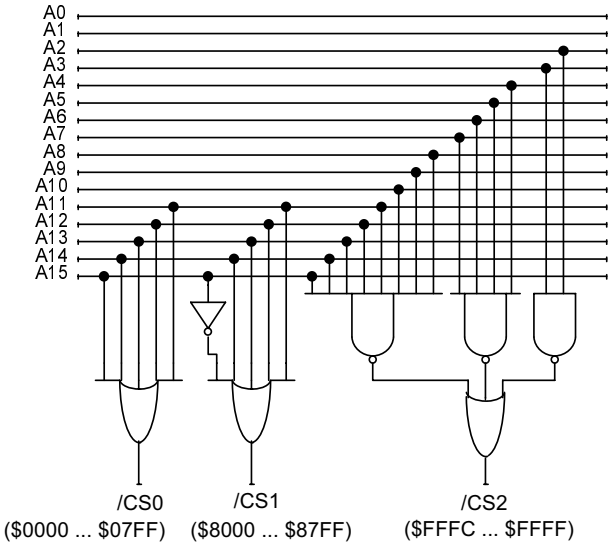
Na desni strani gornje slike je zapis mejnih naslovov v dvojiškem načinu; najvišji (levi) bit je A_{15}, najnižji (desni) pa A_0. Z | so razdeljeni na (levi) zunanji in notranji del. Poiščimo bite, po katerih se zunanji naslovi posameznih področij ločijo med seboj! Vidimo, da ima le prvo področje (ROM) najvišji bit v naslovu A_{15} enak 0, spodnji pa obe 1. Slednji se med seboj ločita po drugem bitu A_{14}.



Naslovni dekodirnik za področje ROM je zato trivialen, oziroma ga sploh ni: kadar je A_{15} = 0, je vedno izbran ROM s signalom $\overline{CS0}$. Kadar pa je A_{15} ena, bo izbrano eno od spodnjih področij: če bo A_{14} nič, bo izbran RAM ($\overline{CS1}$), sicer pa periferni vmesnik ($\overline{CS2}$).

Natančnejša analiza pokaže, da ta dekodirnik generira izbirne signale razen na zahtevanih naslovnih področjih še na nekaterih drugih, npr. ob naslovu $ 0800 se generira signal $\overline{CS0}$, čeprav tam ni pomnilnika. Naslovi se torej celica z notranjim naslovom % 000 0000 0000 v ROMu. Ali natančneje, vsak naslov, ki se pojavi na vodilu, naslovi nek pomnilnik. Čeprav ta situacija na pogled izgleda narobe, ni problematična: če se zavedamo, koliko pomnilnika in na katerih naslovih imamo na voljo, pač ne naslavljamo pomnilnika tam, kjer ga ni.

Takšno naslavljanje imenujemo nepopolno, saj posamezni čipi niso enoumno naslovljeni. Nepopolno naslavljanje ima svoje slabe (ne moremo neposredno vključiti novega pomnilnika, možnost napačnega naslavljanja), pa tudi dobre strani (preprosti dekodirnik z malo elementi).



Na tej sliki je prikazan popoln dekodirnik za zgornjo konfiguracijo. Opazimo, da je neprimerno bolj zahteven kot nepopolni.
Pravili pri naslovnih dekodirnikih sta naslednji:
– vsaka celica pomnilnika mora imeti vsaj en naslov (sicer je ne moremo nasloviti) in
– na vsakem naslovu je lahko kvečjemu ena celica (sicer jih ne moremo ločiti med seboj; naslovi so lahko tudi prazni).

5.5 Upravljanje s pomnilnikom

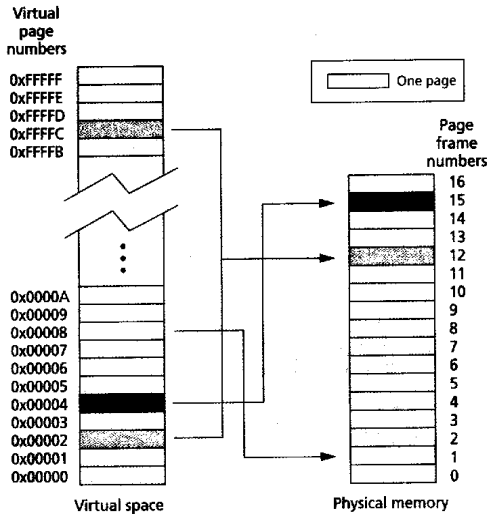
V kompleksnejših mikroračunalnikih je ugodno posebej poskrbeti za upravljanje s pomnilnikom (Memory Management). To porabniku omogoča preslikavo med njegovimi logičnimi naslovnimi področji in fizično shrambo v pomnilniku ter zaščito podatkov na teh področjih preko dodeljenih atributov.

Posebej pomembna funkcija upravljalca pomnilnika je vzdrževanje navideznega pomnilnika. Navidezni pomnilnik je bil razvit z namenom, da bi avtomatiziral premeščanje programov in podatkov med hitrim pomnilnikom in zunanjo shrambo, kadar prvega ni dovolj na razpolago. S tem je ustvarjen občutek velikega enovitega pomnilnika.

Princip delovanja je v tem, da se v fizičnem pomnilniku vzdržujejo tisti podatki, do katerih v nekem trenutku v resnici dostopamo. Tisti, ki jih takrat ne potrebujemo, so medtem spravljeni na disku. Ko pride do preklopa konteksta, se slednji naložijo v prosti fizični pomnilnik, če pa ga ni, spravimo tiste, ki jih najdlje nismo uporabljali, na disk in jih preložimo z aktualnimi.

Strojna oprema in operacijski sistem omogočata, da se preslikave in prelaganje področij pomnilnika izvajajo sproti med tem, ko procesor posega na svoje logično naslovno področje. Preslikava se izvaja po straneh, najmanjših enotah, s katerimi lahko manipuliramo.

Virtualni naslovni prostor, ki ga naslavlja uporabnik, se deli na virtualne strani, ki imajo vsaka svoj naslov. Fizični pomnilnik je razdeljen na enako velike naslovljene okvirje. Najpreprosteje rečeno je torej virtualno naslavljanje preslikava virtualnih naslovov na fizične okvirje.



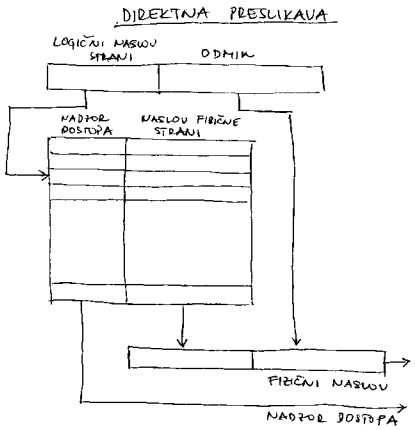
Informacije o preslikavah so zbrane v tabeli strani (page table); te so sestavljene iz posameznih vpisov tabele strani (page table entries, PTE), ki se nanašajo na posamezne strani. Vsak vpis (PTE) vsebuje vsaj podatke o virtualnem in fizičnem naslovu v pomnilniku ali na disku, lahko pa tudi dodatne informacije, ki služijo za zaščito podatkov, na primer, ali je na stran mogoče vpisovati podatke, ali se na njej nahaja program, ipd. V njih je spravljena tudi informacija o tem, kdaj je bila stran nazadnje uporabljena. To informacijo upošteva sistem, ko se odloča, katere strani v fizičnem pomnilniku bo prekril z zahtevanimi, potem, ko je v njem zmanjkalo prostora.

Nekaterih podatkov nam ni treba eksplicitno vzdrževati v tabeli; z ustrezno organizacijo jih lahko implicitno ugotovimo iz položaja v tabeli.

Za hitrejše delovanje preslikave večina sodobnih sistemov uporablja hitre pred-

pomnilnike (translation lookaside buffer, TLB), v katerih vzdržuje najbolj sveže podatke o zadnjih preslikavah. Če ob posegu v virtualno stran njen fizični naslov najdemo v TLB, ga takoj uporabimo. Kadar pa ga ni, ga moramo ugotoviti iz tabel.

V preteklosti, ko so bili pomnilniki manjši, pa tudi v preprostejših sodobnih sistemih, se je tabela strani nahajala v t.i. direktni tabeli (direct table). V direktni tabeli strani so zvezno navedeni podatki za vse virtualne strani. Dokler je ta razmeroma majhna, je upravljanje s pomnilnikom lahko izvedeno neposredno v strojni opremi.



Logični naslov je razdeljen na dva dela, na naslov virtualne strani in na odmik (notranji naslov) na strani. Naslov strani izbere vpis v tabeli strani (PTE), ki vsebuje atribute te strani za nadzor dostopa in naslov fizične strani. Iz slednjega in iz odmika se sestavi fizični naslov podatka v pomnilniku, iz atributov pa se razbere, ali je ta stran na razpolago, ali jo je treba naložiti, ter privilegije, potrebne za njen dostop.

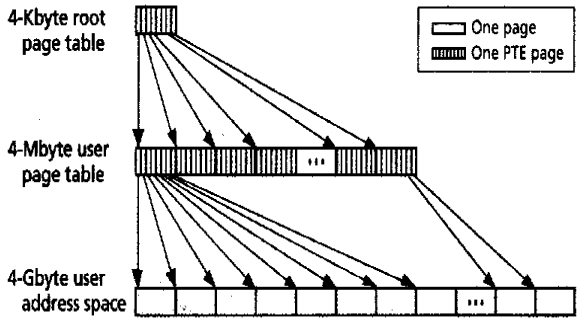
S povečevanjem pomnilniškega prostora je bilo potrebno to tabelo preseliti v zunanji pomnilnik, njeno obdelavo pa izvesti s programiranjem; upravljanje s pomnilnikom postane funkcija operacijskega sistema. To pa pomeni, da je izvedba te funkcije bistveno počasnejša.

Tabele strani sedaj ne morejo več vsebovati podatke o vseh virtualnih straneh, ker jih je preveč; to pa pomeni, da naslov strani ne določa več pozicije vnosa v tabeli strani (PTE). Podatke o preslikavi je zato potrebno poiskati v teh tabelah.

Preiskovanje velikih tabel je posebej časovno zahtevna funkcija, zato so bili razviti posebni algoritmi. Običajni pristopi upoštevajo hierarhično zgradbo oz. segmentacijo pomnilnika: ta je razdeljen na segmente, ti pa dalje na strani. Preslikava sedaj poteka v dveh korakih: najprej se poišče naslov fizičnega segmenta, nato pa naslov strani znotraj njega. Ker je segmentov bistveno manj kot strani, znotraj segmentov pa spet bistveno manj strani, kot jih je vseh v pomnilniku, je iskanje preslikav precej hitrejše.

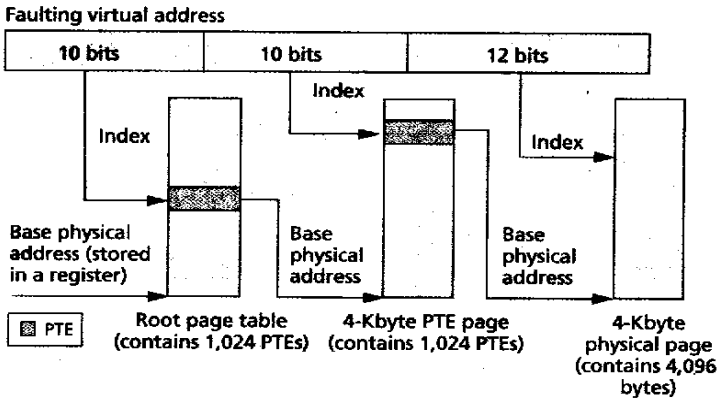
Slika prikazuje naslednji primer: pri 32-bitnem naslavljanju, ki omogoča 4 Gbyte (2^{32}) prostora imamo pomnilnik organizran po 4kbytnih (2^{12}) straneh. Takšnih strani je torej 1M ($2^{32}/2^{12} = 2^{20}$). Za preiskovanje takšne tabele potrebujemo veliko časa. Zato virtualni naslov strani sestavimo iz dveh delov po 10 bitov: prvi določa vnos v korenski tabeli strani dolžine 4-Kbyte, ki podaja začetek uporabniške tabele strani, drugi del naslova pa iz slednje tabele poišče fizični naslov strani. Na tej strani velikosti 4kbyte s pomočjo 12-bitnega notranjega naslova določimo končni fizični naslov.

Korenska tabela strani je pri tem kratka in zato brez večje izgube vedno prisotna v pomnilniku. Od ostalih tabel pa so prisotne le tiste, ki so v resnici uporabljene. Običajno tudi pripadajo enemu opravilu; fizični okvirji, na katere preslikujejo virtualne naslove, morajo običajno biti istočasno prisotni v pomnilniku, kar tudi poenostavi mehanizem prenosa fizičnih strani med diskom in pomnilnikom: podatek o razpoložljivosti podatkov v okvirju fizičnega pomnilnika je lahko vpisan že v korenski tabeli preslikav, med diskom in pomnilnikom pa lahko naenkrat prenesemo večjo količino pomnilnika.



Na spodnji sliki je za ta primer prikazan postopek sprehoda po tabeli (tablewalking); uporabljen je način preiskovanja od zgoraj navzdol (Top-down traversal,

Forward-mapped page table):



poleg tega obstajata še sodobnejša načina preiskovanja od spodaj navzgor in inverzne preslikave.

Poglavje 6

Primer zgradbe hipotetičnega mikroračunalnika

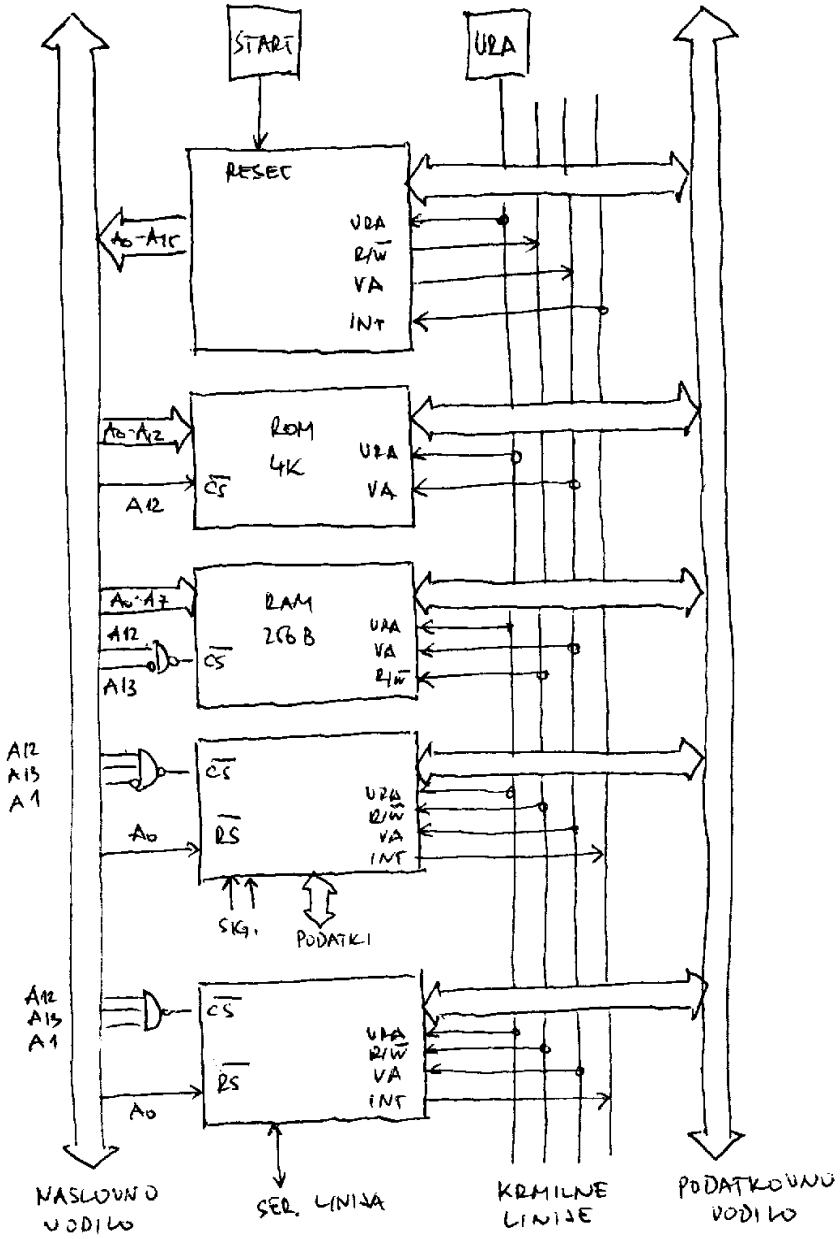
Primer vezave mikroračunalnika s sinhronim protokolom prenosa podatkov

ura daje takt,
z VA procesor pove, da gre za cikel prenosa podatkov,
z R/W njegovo smer,
z INT pa periferija proži prekinitve na procesorju.

Pomnilniška slika:

```
ROM   -      0 -  FFF    YY00 XXXX XXXX XXXX
RAM   -   1000 - 10FF    YY01 YYYY XXXX XXXX
PAR.  -   3000 - 3001    YY11 YYYY YYYY YY0X
SER.  -   3002 - 3002    YY11 YYYY YYYY YY1X


(Y - redundantni biti v nepopolnem naslavljanju,
 X - notranji naslov v naslovnem področju)
```

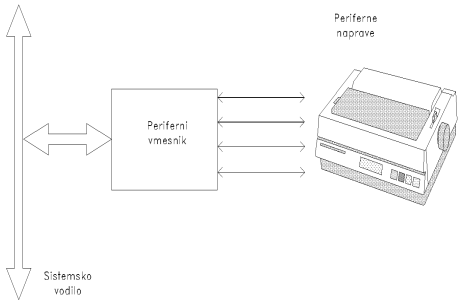

START
URA
RESET
A0-A15
URA
R/W
VA
INT
A0-A12
ROM
4K
CS
A12
URA
VA
A0-A7
A12
A13
RAM
2GB
CS
URA
VA
R/W
A12
A13
A1
CS
RS
A0
URA
R/W
VA
INT
SIG.
PODATKI
A12
A13
A1
CS
RS
A0
URA
R/W
VA
INT
NASLOVNO VODILO
SER. LINIJA
KRMILNE LINIJE
PODATKOVNO VODILO

Poglavje 7

Povezava mikroprocesorja z okolico

Vhodno/izhodne (periferne, V/I, Input/output, I/O,) enote so namenjene povezovanju mikroprocesorskega sistema z okoljem. Mednje sodijo po eni strani standardne naprave, kot so na primer diskovne enote, terminali, tiskalniki in podobno, ter senzorji in aktuatorji za vodenje procesov po drugi strani.

Da bi lahko periferne enote, ki so pogosto samostojne naprave, vodili z mikroračunalnikom, jih moramo nanj priključiti s *perifernim vmesnikom* (Peripheral Interface). Mikroprocesor pri delovanju ne vidi resnične periferne naprave, temveč le njen vmesnik, katerega programski model jo identificira.



Naloga vmesnika na periferni strani je generirati krmilne signale ter signale za

prenos podatkov na in s periferne enote. S tem pred programerjem skrije podrobnosti in omogoči delo z abstraktnim programskim modelom.

V primeru, prikazanem na prejšnji sliki, programer vpiše niz znakov, ki jih želi izpisati, zaporedoma v vmesnikov register, ki logično predstavlja izhod na tiskalnik. Sam generira fizične signale za izbiro tiskalnika, izvede morebitne pretvorbe, prenos podatkov, sinhronizacijo delovanja ipd.. ter to ponavlja, dokler niso izpisani vsi podatki iz vmesnega pomnilnika.

7.1 Prenos podatkov med V/I vmesniki in mikroprocesorjem

Obstajata dva principa priključevanja perifernih vmesnikov: v *ločen periferni prostor* in v *pomnilniško preslikan periferni prostor* (Memory-mapped I/O).

Pri *pomnilniško preslikanem perifernem prostoru* se registri perifernih vmesnikov obnašajo kot lokacije pomnilnika, zato je tudi njihovo prepoznavanje podobno kot pri pomnilniku. Možni so vsi načini naslavljanja, ki omogočajo elegantno programiranje.

Periferni vmesniki običajno zasedajo malo naslovov, kar pomeni, da moramo za popolno dekodiranje upoštevati mnogo naslovnih linij (glej poglavje 5.4). Naslove perifernih vmesnikov zato praviloma nepopolno dekodiramo. Slabost tega načina pa je, da razdrobi celovitost pomnilniškega naslovnega prostora. Na ta način je periferija priključena npr. na mikroprocesorje pri Motorolinih družinah.

Drugi način je priključitev perifernih vmesnikov v ločenem, *perifernem naslovnem prostoru*, kot je uporabljen na primer pri mikroprocesorjih iz Intelovih družin. Poleg naslova, ki je običajno mnogo krajši kot v pomnilniško preslikanem načinu, pošlje mikroprocesor po posebnih statusnih linijah še signal, da gre za komunikacijo s periferijo. Za programiranje imamo na razpolago posebne ukaze, običajno *IN* in *OUT*. Načini naslavljanja so zelo omejeni, programiranje je manj elegantno, vendar so dekodirniki za periferne vmesnike preprostejši, pa tudi pomnilniški prostor je nerazdrobljen. Seveda lahko pri procesorjih s tem načinom priključevanja alternativno uporabimo tudi preslikavo perifernega v pomnilniški prostor.

Periferne enote, posebno tiste z mnogo mehanike, so praviloma za nekaj razre-

dov počasnejše od mikroprocesorja, ki bi zaradi tega moral pri prenosu podatkov čakati nanje. Zato je ugodno, da ima vmesnik vgrajen lokalni pomnilnik, kamor lahko procesor v enem kosu prenese podatke, ki se nato postopoma pošiljajo periferni enoti.

Komunikacija, na primer, je praviloma počasen postopek; če pogosto prenašamo bloke podatkov, je smiselno v pripadajoči periferni vmesnik vgraditi lokalni pomnilnik. Procesor prenese podatke vanj, nato pa nadaljuje z drugim delom, medtem ko jih vmesnik sam postopoma pošilja prejemniku.

7.2 V/I naprave in V/I vmesniki

Periferne vmesnike delimo na *univerzalne* in *namenske*.

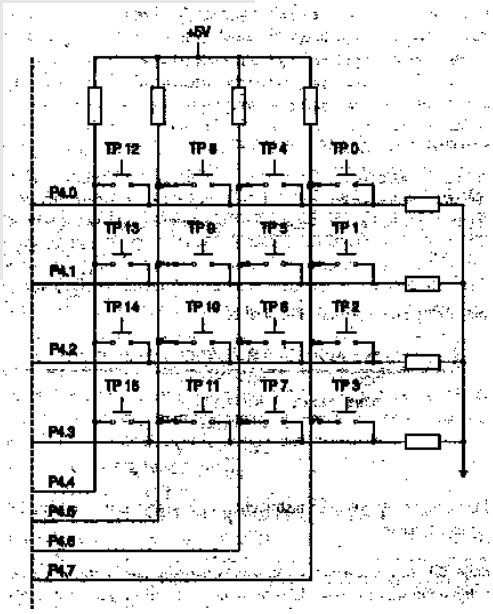
- *univerzalni* vmesniki so tisti, ki omogočajo priključitev signalov z najrazličnejših naprav. Poznamo:
  - paralelne vmesnike, preko katerih lahko na mikroprocesorski sistem priključimo digitalne vhodne in izhodne linije;
  - serijske vmesnike, ki služijo za priključitev serijskih kanalov za prenos podatkov (npr. na terminale ipd.);
  - analogno/digitalne in digitalno/analogne vmesnike za priključevanje analognih veličin;
  - števce in časovna vezja (Timer), ki merijo čas ter štejejo dogodke
  - in druge.
- *namenski* vmesniki so namenjeni uporabi v točno določenih aplikacijah, npr. krmilnik diskov in disketnikov, krmilnik zaslonov, tipkovnic, komunikacijski vmesniki za različne protokole in podobno. Univerzalni vmesniki so praviloma preprostejši od namenskih.

7.2.1 Paralelni vmesniki

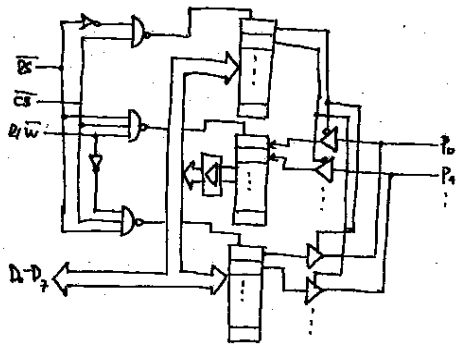
Najpreprosteši vmesniki med mikroprocesorjem in periferijo. Vmesnik med logičnimi(programiranimi podatki) in fizičnimi signali - linijami kot izhodi iz mikroračunalnika.

Registri - krmilni, statusni (lahko kombinirani, če je bitov malo) in podatkovni. Kar se vpiše v register, se pojavi kot napetost na izhodu in obratno: priključeni napetostni signali so v obliki bitov dostopni v podatkovnem registru.

Uporaba paralelnega vmesnika: tipkovnica. Zmanjševanje števila linij (namesto linije z vsake tipke matrična povezava.



Izvedba hipotetičnega paralelnega vmesnika. 8 digitalnih vhodno - izhodnih linij; en podatkovni register, en smerni register: enica v njem pomeni izhod iz vmesnika, ničla vhod.
Krmilni signal $\overline{CS}$ pogojuje, da bo izbran periferni vmesnik. Med smernim in podatkovnim registrom izbira krmilni signal $\overline{RS}$, na katerega bo najpogosteje priključena naslovna linija A_0: sodi naslovi izberejo smerni, lihi pa podatkovni register.
Registri so izvedeni z zadrževalniki (zapah, latch): ti signale prepuščajo, dokler je njihov krmilni signal 1 oziroma zadržijo zadnje stanje, ko je preide v 0.
V resnici je podatkovni register sestavljen iz dveh zadrževalnikov: en deluje kot vhod, drugi pa kot izhod. Tisti, ki podatke pošilja na podatkovno vodilo, je od njega ločen še z elektronskimi stikali (digitalnimi vmesniki, bufferji), da njegove vsebine ne čutimo na vodilu, kadar ni izbran.



Vsaka digitalna V/I linija je priklopljena na vhod enega in izhod drugega vmesnika: glede na stanje pripradajočega bita v smernem registru torej deluje kot vhod ali izhod:

- digitalni izhodi: podatek se s podatkovnega vodila prenese v zapah in se tam zadrži. Podatki iz bitov, ki so predvideni za izhode, se preko vmesnikov, ki jih odpirajo biti smernega registra, prenesejo na izhodne sponke.
- digitalni vhodi: vhodni signali, ki so omogočeni z biti smernega registra, so pripeljani na zapah. V trenutku čitanja se zapah zapre, s čemer se za ta čas ustavi vpisovanje vanj in se odprejo vmesniki na podatkovno vodilo.

Seveda se npr. v primeru čitanja prenese celotni byte na podatkovno vodilo; biti, ki so deklarirani kot izhodi, pri tem nimajo smiselne vrednosti in jih moramo ignorirati. Podobno velja pri pisanju.

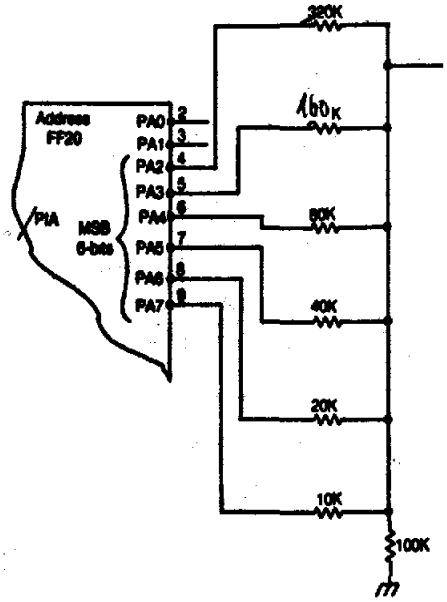
7.2.2 Serijski vmesniki

Osnova: paralelno/serijska oz. serijsko/paralelna pretvorba.

RS232 protokol: star, v uporabi za preprostejšo in praviloma počasnejšo standardno periferijo, kot so terminali, miške, modemi.
V novejšem času so aktualni distribuirani sistemi: sistemi, pri katerih je procesna moč vgrajena tam, kjer je potrebna: inteligentni senzorji in aktuatorji, ipd. Te aktivne ("inteligentne") komponente so povezane v sistem s serijskimi vodili, kot so I^2C, CAN, Interbus-S, itd. Serijska vodila so praviloma hiotrejša, mnogo zmogljivejša in nudijo več možnosti kot RS232. Periferni vmesniki zanje so vedno pogosteje vgrajeni v mikrokrmilnike ali pa so na voljo v preprostih namenskih čipih.

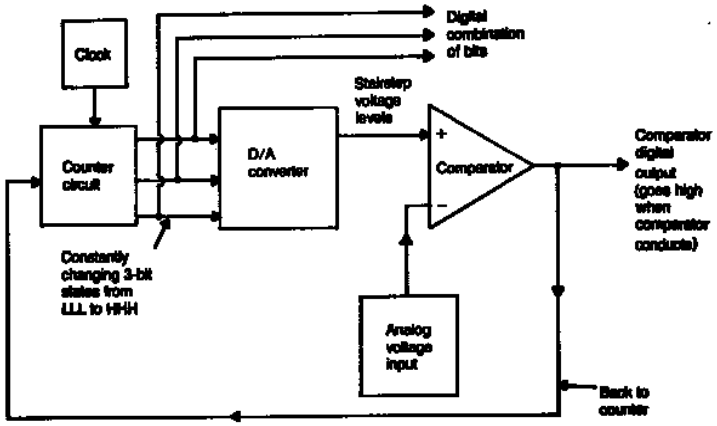
7.2.3 Analogni vmesniki

D/A: najpreprostejši princip:



Ni praktično uporaben. Namesto njega R/2R vezje.

A/D: en od preprostejših načinov, na osnovi D/A pretvornika



7.2.4 Časovniki, števci

7.3 Neposredni dostop do pomnilnika

Prenos podatkov med mikroprocesorjem in perifernimi vmesniki v splošnem poteka po enakem protokolu kot komunikacija s pomnilnikom. Z ozirom na količino podatkov, ki jih je potrebno prenesti, lahko uporabimo posebne tehnike, kot je neposredni dostop do pomnilnika.

Univerzalni mikroprocesorji morajo za prenos vsakega podatka iz bloka izvršiti cel program:

prečitaj podatek z vhodne enote,
vpiši podatek na izhodno enoto,
zmanjšaj števec podatkov, ki jih je treba še prenesti,
preveri, ali se je iztekel,
če ne, povečaj naslove na vhodni in izhodni enoti in
ponovi cikel.

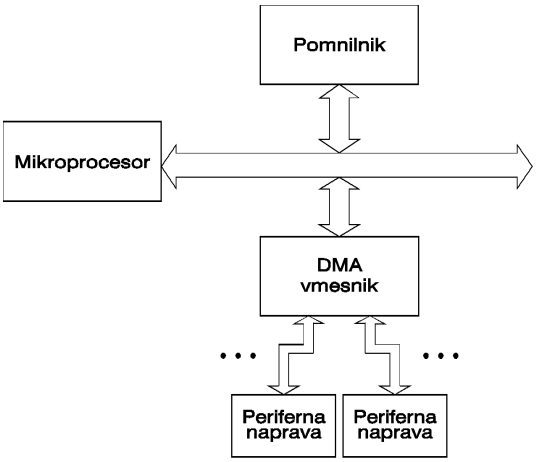
Zaradi von Neumannove organizacije je seveda treba tudi vsakega od ukazov prečitati iz pomnilnika (glej poglavje **??**). Prenos velikih blokov podatkov je torej

časovno zahtevna operacija.

Prenos podatkov s perifernih enot in nanje je pogosto ugodno organizirati v velikih paketih, saj je čas za fizični dostop do področja, kjer se na mediju nahajajo, mnogokrat bistveno večji kot za čitanje in prenos zaporednih podatkov (npr. pri komunikaciji z diski). Tukaj se srečamo z zgoraj opisanim problemom, ki ga rešujemo s tehniko *neposrednega dostopa do pomnilnika*, (DMA, Direct Memory Access).

Pri tej tehniki uporabimo posebno enoto, *DMA krmilnik*, ki na zahtevo zmore kandidirati za vodilo in ga upravljati ter prenašati podatke. To je tudi njegova edina funkcija, pripadajoči "program" pa je implementiran v aparaturni opremi. Potrebni podatki, naslov izvora, cilja ter število besed za prenos se vpišejo v registre krmilnika. Prenos se izvaja bistveno hitreje kot s programom na univerzalnem procesorju, saj ni potrebno prenašati ukazov iz pomnilnika, nekatere funkcije, kot so dekrementiranje števcev in inkrementiranje naslovov, pa se izvedejo na strojni opremi paralelno s prenosom podatkov.

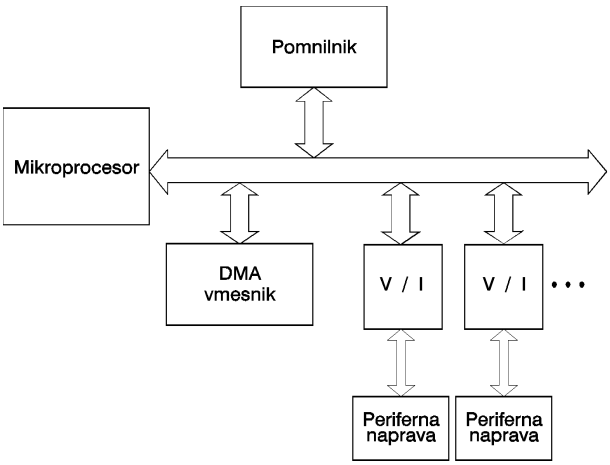
Na spodnji sliki je prikazana običajna priključitev periferne naprave preko DMA krmilnika.



V času delovanja mikroprocesorja jo povezuje v sistem V/I vmesnik, DMA krmilnik je pasiven. Ko pa ta dobi ukaz za prenos podatkov, kandidira za vodilo. Ko ga dobi, se procesor odklopi od sistema in protokol vodi DMA krmilnik. Podatki se pretakajo z vhodne naprave na vodilo preko njega, zaradičesar se ta način tudi

imenuje *pretočni* (flow–through).

Poleg pretočnega poznamo tudi *leteči* (fly-by ali floating) način priključitve periferne enote preko neposrednega dostopa do pomnilnika. Pri njem se podatki pretakajo neposredno med periferno napravo in pomnilnikom, DMA krmilnik le skrbi za generiranje potrebnih signalov. Podatki se pri tem pretakajo po sistemskem vodilu. Nekatere izvedbe imajo med pomnilnikom in perifernim vmesnikom posebno dodatno pomnilniško vodilo, ki je včasih tudi serijsko, in ki omogoča prenos podatkov paralelno z normalnim delovanjem ostalega dela sistema. Pri letečem načinu lahko en DMA krmilnik streže tudi več V/I enot, ker te niso neposredno priključene nanj.



Prenos podatkov poteka iz *izvora* v *ponor*. Pot, ki se pri tem ustvari, vključno z generacijo nadzornih signalov in s pomožnimi strukturami, kot so števci, registri ipd., se imenuje kanal. DMA krmilniki imajo običajno več kanalov, kar omogoča komuniciranje z več perifernimi enotami, vendar ne istočasno.

Področja uporabe DMA so povsod, kjer je treba prenesti veliko število zaporednih podatkov (diski, osveževanje ekranov ipd.). Poseben problem predstavlja tudi shranjevanje podatkov, ki prihajajo v periferno enoto tako hitro, da jih s programom ne uspemo prečitati. Tukaj je neposredni dostop do pomnilnika celo edina rešitev.

Slabost te tehnike je, da zahteva dodatno aparaturno opremo in poseg v arhitekturo sistema (pri dodeljevanju vodila). Med prenosom podatkov je delovanje mikro-

procesorja moteno ali celo ustavljeno, glede na izvedbo.

7.3.1 Vrste DMA prenosa

Metode za prenos podatkov preko neposrednega dostopa do pomnilnika delimo na dve veliki skupini:

- blokovni prenos in
- besedni prenos s krajo ciklov (cycle stealing)

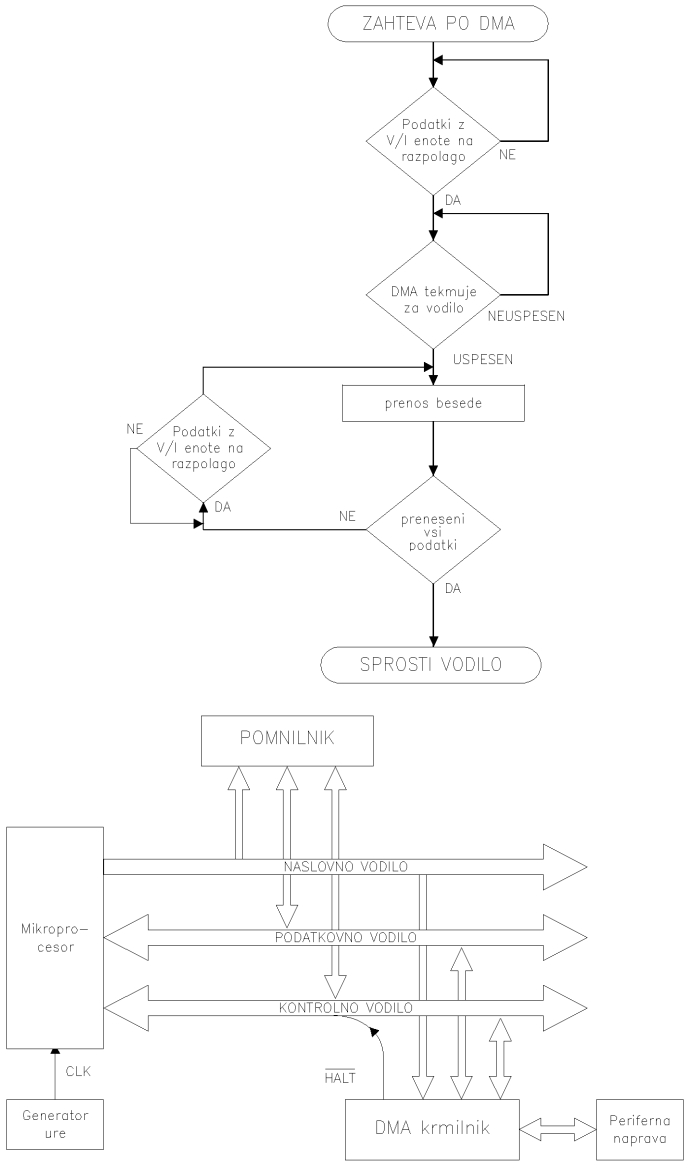
ter kompromisno metodo

- eksplozijski prenos (burst mode)

7.3.1.1 Blokovni prenos

Pri blokovnem prenosu se prenese blok podatkov poljubne dolžine v enem kosu, ko je DMA krmilnik gospodar vodila. Ta način zagotavlja največjo možno hitrost prenosa podatkov. Težava je v tem, da je med prenosom mikroprocesor odklopljen od vodila in je njegovo delovanje zadržano. To pa lahko pomeni resne probleme pri sistemih, ki morajo hitro reagirati na signale iz okolja, na primer alarme ipd.

Ko je periferna naprava pripravljena za prenos podatkov, DMA krmilnik prevzame vodilo in ga sprosti šele, ko so prenešeni vsi podatki. Pri tem računamo na to, da so podatki že na razpolago v periferni napravi, oziroma da bodo sproti in pravočasno prihajali. Če podatkov ni ali če kak člen v prenosni verigi odpove, pomeni to izpad sistema.

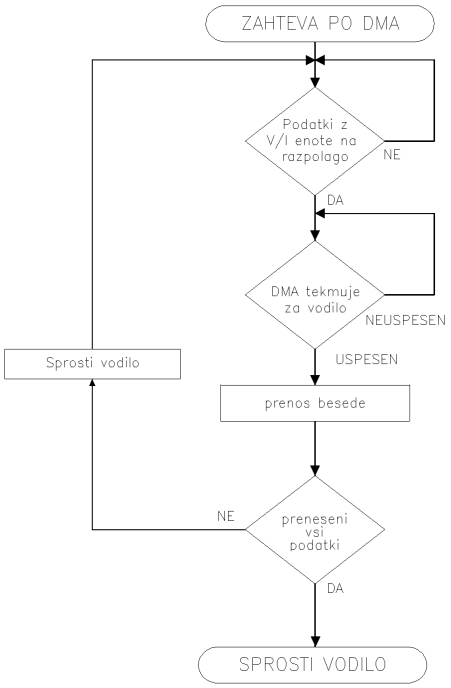


Na gornji sliki je prikazana možna realizacija takšnega prenosa podatkov preko neposrednega dostopa do pomnilnika v blokovnem načinu. Ko dobi DMA krmilnik ukaz za prenos podatkov, aktivira signal $\overline{HALT}$. S tem prisili mikroprocesor, da zamrzne delovanje, postavi svoje priključke na vodilo v stanje visoke uporno-

sti in se s tem umakne z njega. Ko DMA krmilnik konča prenos, umakne signal $\overline{HALT}$ in s tem reaktivira delovanje mikroprocesorja.

7.3.1.2 Prenos s krajo ciklov

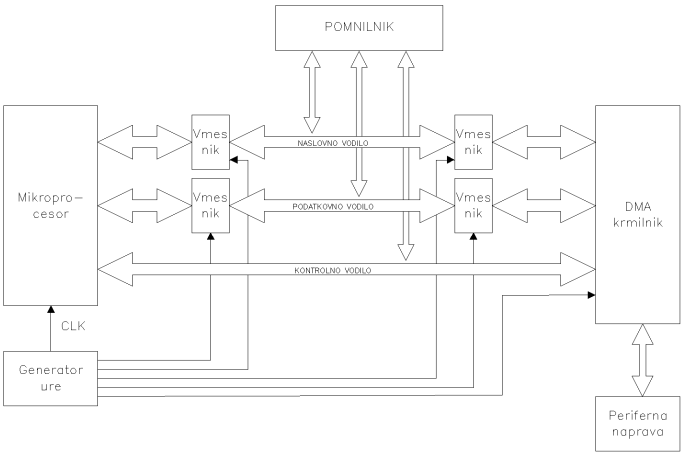
Pri tej metodi DMA krmilnik "krade" po enega ali nekaj ciklov za prenos podatkov po vodilu, po zaključku pa vodilo obvezno vrne mikroprocesorju. Prenos bloka podatkov je torej prekinjan s procesorskimi cikli, oziroma delovanje mikroprocesorja prekinja DMA krmilnik. Delovanje je prikazano na spodnji sliki.



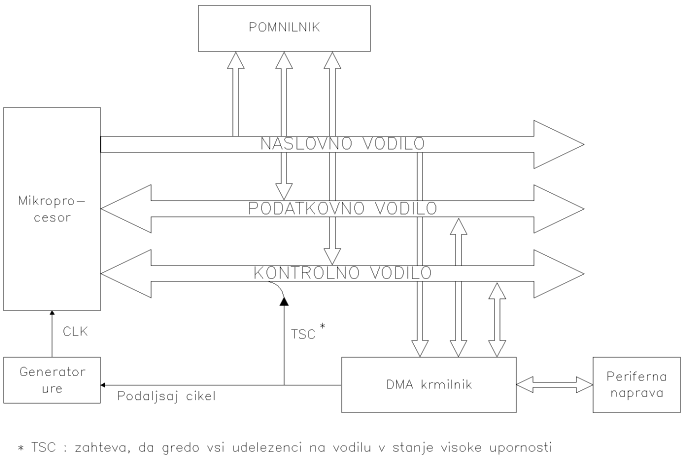
Ko je na periferni napravi pripravljena beseda za prenos, si DMA krmilnik pridobi pravico za prenos besede. Po prenosu prepusti vodilo mikroprocesorju.

Ukradeni so lahko celotni ali pa samo deli procesorskih ciklov, v odvisnosti od izvedbe. Pri mikroprocesorjih, ki dostopajo do vodila samo v vnaprej določenih delih svojih strojnih ciklov, lahko za DMA prenos uporabimo ostanek le-teh. Podobna situacija je, če je cikel mikroprocesorja precej večji od cikla vodila; takrat

lahko vodilo preklapljamo ali *multipleksiramo* med mikroprocesorjem in DMA krmilnikom. Za to je potrebna posebna aparaturna realizacija, prikazana na spodnji sliki. To je način, ki ne zavira zmogljivosti mikroprocesorja in ga imenujemo tudi *transparentni* način.



Ura pri tem z enkrat višjo frekvenco, kot jo dovoljuje procesor, preklaplja vmesnike, s katerimi “odreže” od vodila v enem trenutku mikroprocesor, v drugem pa DMA krmilnik.

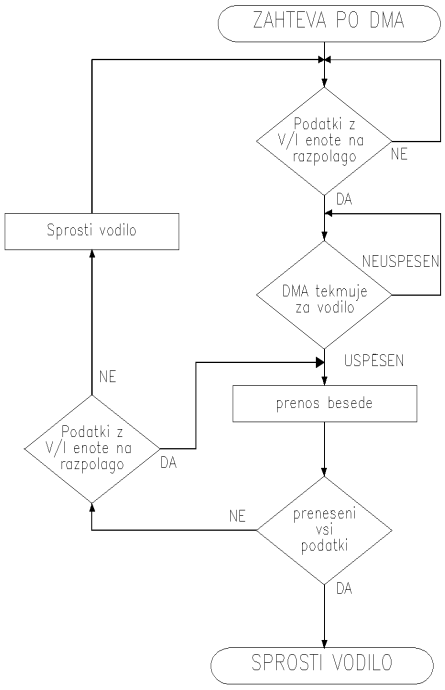


Kadar pa tak način priključitve ni mogoč, lahko uporabimo tehniko podaljševanja mikroprocesorskega cikla, ki je prikazana na gornji sliki. Ob zahtevi za prenos

DMA krmilnik vsak mikroprocesorski cikel podaljša še za en svoj cikel in pošlje eno besedo. V ta namen generira signal, ki da ukaz generatorju ure in obenem zahteva odklop vseh ostalih enot od vodila.

Kraja ciklov je razmeroma počasen način prenosa podatkov neposredno v pomnilnik, vendar je tudi vpliv na delovanje mikroprocesorja mnogo manjši in manj nevaren kot pri blokovnem prenosu.

7.3.1.3 Eksplozijski prenos



Kadar želimo visoko hitrost prenosa podatkov, pa vendar možnost dovolj hitrega reagiranja na signale iz okolice, uporabimo kompromisno rešitev, eksplozijski prenos, ki je prikazan na gornji sliki. Pri tem načinu DMA krmilnik prenaša podatke, dokler je periferna naprava pripravljena za komunikacijo. Čim pa bi bilo treba zadrževati prenos zaradi periferne naprave, prepusti vodilo mikroprocesorju. Ker je sam prenos pripravljenih podatkov razmeroma hiter, je procesor na ta način zadržan minimalni čas pri maksimalni hitrosti prenosa. Ta način prenosa se

običajno izvede z ustavljanjem procesorja.

Posebej primeren je ta način prenosa pri komunikaciji z diskom. Za disk je značilno, da potrebuje mehanizem precej časa, da pozicionira glave in doseže podatke, sam prenos pa gre relativno hitro. Zato je ugodno, če DMA zavzame vodilo šele, ko so podatki res že na voljo.

7.4 Upravljanje z V/I napravami

7.5 Inicializacija

Periferne vmesnike in enote je praviloma potrebno ob zagonu sistema (včasih pa tudi med delovanjem) inicializirati. To lahko izvedemo z generiranjem signala $\overline{RESET}$, kadar sistem to omogoča (npr. Motorolini mikroprocesorji iz družine 68K), ali pa s posebnimi ukazi.

Pri serijski komunikaciji se rado zgodi, da sistem "obvisi" v t.i. smrtnem objemu: zaradi asinhronizma partnerja v komunikaciji čakata drug na drugega. Te situacije praviloma ne moremo rešiti drugače kot z re-inicializacijo.

Ob inicializaciji se postavijo registri in podsistemi v začetno stanje. Druga pomembna funkcija inicializacije je obveščanje perifernega vmesnika o prekinitvenem vektorju: pri vektorskih prekinitvah prekinjevalec javlja procesorju številko vektorja izjeme, ki se naj izvede. To pa mu mora programer najprej "sporočiti".

7.6 Sinhronizacija in nadzor V/I operacij

Periferne enote odražajo obnašanje okolja, ki je običajno asinhrono z delovanjem mikroprocesorskega sistema. Da lahko prepoznamo zahtevo po prenosu podatkov, je potrebno vpeljati sinhronizacijske mehanizme. Delimo jih na dve skupini:

- programirani prenos podatkov in
- prekinitveni prenos podatkov.

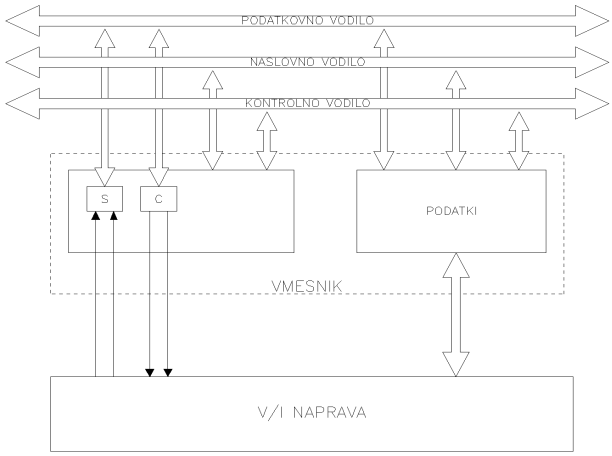
7.6.1 Programirani prenos podatkov

Programirani prenos podatkov je najpreprostejša metoda. Ločimo *brezpogojni* in *pogojni* programirani prenos.

Pri *brezpogojnem* mikroprocesor sinhrono s svojim programom (torej nesinhronizirano z okoljem) prenaša podatke na ali z perifernega vmesnika. Ta način je možen, kadar je pogostost izmenjave podatkov dovolj velika, da so podatko vedno dovolj "sveži", periferna naprava pa ob vsakem trenutku pripravljena komunicirati.

Kadar pa je periferna naprava sposobna izmenjevati podatke samo ob določenih trenutkih, uporabimo *pogojni* programirani prenos. Procesor ciklično tipa status perifernih enot, da bi ugotovil, ali so pripravljene komunicirati oziroma ali so podatki pripravljeni. Zaradi tega se ta način imenuje tudi povpraševalni ali statusno voden V/I *(polling)*.

Ciklično povpraševanje je časovno zahtevna operacija, ki močno vpliva na splošno zmogljivost sistema in na količino podatkov, ki jih zmoremo prenesti. Enota, ki želi prenašati podatke, mora čakati na naslednji ciklus tipanja, da se njena zahteva prepozna. Poleg tega ta postopek zahteva periodično aktivnost procesorja, ki je zato bolj zaseden.



Na gornji sliki je prikazan vmesnik za programirano sihronizacijo V/I operacij. Poleg podatkovnega prenosnega dela vsebuje tudi krmilne in statusne registre.

Periferna naprava zahtevo po prenosu signalizira vmesniku, ki jo kot zastavico v statusnem registru prikaže mikroprocesorju. Ta ciklično čita statusne registre vseh enot, ki lahko zahtevajo prenos, in ga izvaja.

7.6.2 Prekinitveni prenos podatkov

Glavni slabosti programiranega prenosa podatkov, zasedenost procesorja s cikličnim čitanjem ter zakasnitev med pojavom potrebe po prenosu podatkov in zaznavo le-te, sta odpravljeni pri prekinitveni realizaciji prenosa podatkov. Delovanje prekinitev je podrobneje opisano v poglavju **??**.

Ko periferna naprava želi prenašati podatke, to signalizira vmesniku. Ta pošlje mikroprocesorju *prekinitev*. Procesor reagira tako, da prekine obdelavo trenutnega programa in izvede *prekinitveno strežno rutino.* Na ta način je nadzor nad sistemom prevzela periferna naprava, ki sedaj časovno usklajuje dogajanje.

Prekinitveni princip prenosa podatkov pomeni *opazovanje dogodkov v sistemu*, programirani pa *opazovanje stanj*. Kljub vsem dobrim lastnostim prekinitvenega prenosa in slabostim programiranega, le-ta nudi pomembno prednost pri odpornosti na napake *(fault tolerance – trdoživost)*: če se iz kakršnega koli vzroka zgodi napaka in se izgubi prekinitev, pomeni, da smo morda izgubili spremembo stanja, kar je lahko nepopravljiva napaka (npr. števec bo zaostal, stanje sistema se ne bo obrnilo ipd.) Ob napaki pri tipanju stanja bo dogodek gotovo prepoznan, pa čeprav šele v naslednjem ciklusu, saj ga bo ohranil status periferne naprave.

Poglavje 8

Izjeme in prekinitve

8.1 Splošni pojmi

Izjema je *dogodek* (signal, napaka ali ukaz), ki povzroči, da se normalni način delovanja – izvajanja ukazov iz programa – prekine in izvede *strežni program izjeme*, ki ga je programer predvidel kot reakcijo nanjo. Ko se ta program zaključi, se glavni potek izvajanja nadaljuje tam, kjer je bil prekinjen. Postopek ko en program začasno prekinemo in ga nadomestimo z drugim se imenuje *zamenjava konteksta*[1] (vsebine). Običajno zahtevamo, da prekinjeni program (razen časovnega zamika) ne čuti, da se je vmes izvajala strežba izjeme.

Viri za izjeme so lahko *znotraj* ali *zunaj* mikroprocesorja. Notranji viri izjem so na primer *ukazi* (TRAP, DIV..), posebni načini delovanja mikroprocesorja (npr. sledenje[2]) ali *napake* (napake v naslavljanju, nedovoljeni ukazi, prekoračitev privilegijev ipd.) Zunanji viri izjem pa so prekinitve, signali za razne napake, signal RESET ipd.

Razlikujemo dve kategoriji izjem:

- *pasti:* pasti so običajno notranje generirane izjeme, ki prestrezajo posebne dogodke v delovanju sistema. Ti so lahko nepričakovani in nezaželeni, kot

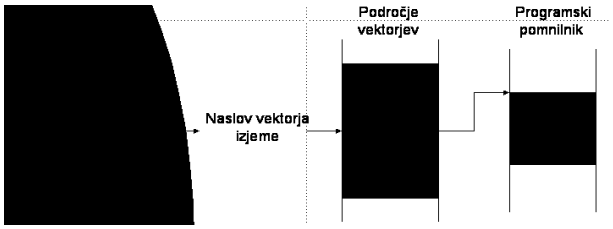
[1] Context switch
[2] Trace mode

so npr. deljenje z 0 ali napaka v naslovu; sem spada tudi napaka na vodilu, ki je sicer zunanja izjema. Med pasti prištevamo še sledenje, pri katerem se delovanje programa prestreže po vsakem ukazu.

- *prekinitve:* običajno jih generirajo periferne naprave, ko želijo, da mikroprocesor reagira na nek dogodek iz okolice (npr. ob prispetju nekega podatka, po koncu pošiljanja nekega podatka, ipd.). Uporaba prekinitev je dodatno obdelana v poglavju o sinhronizaciji prenosa podatkov s perifernih naprav 7.6.2.

Zgoraj omenjeni *strežni program izjeme* je program, ki se naj izvede ob pojavu izjeme. Običajno je *rezidenčno* (stalno) naložen v pomnilniku in se pri večini mikroprocesorjev požene v posebnem (nadzornem) načinu delovanja. Ker je virov izjem lahko več, lahko nastopa tudi več strežnih rutin. Vsaka izmed rutin ima svoj začetni naslov, v mikroprocesorju pa mora obstajati mehanizem, preko katerega se izbere ustrezna strežna rutina. To lahko naredi na dva načina:

- *Naslov strežne rutine se določi neposredno.* Vsaka izjema oz. skupina izjem ima vnaprej določeno lokacijo v kodi na katero se prenese izvajanje programa ob izjemi. To pomeni, da se v programski števec vpiše neka vnaprej znana vrednost. Programer mora zagotoviti, da bo začetna lokacija rutine za strežbo izjeme enaka predpisani.
- *Naslov strežne rutine se določi posredno.* V tem primeru se naslovi strežnih rutin nahajajo na vnaprej znanih pomnilniških lokacijah. Eno izmed teh lokacij običajno imenujemo *vektor izjeme*. Ob izjemi mikroprocesor prečita vsebino vektorja izjeme in ga prenese v programski števec. Programer pripravi program za strežbo izjeme in njegov začetni naslov vpiše v vektor.



Naslove vektorjev izjem običajno določi proizvajalec mikroprocesorja. Pri nekaterih mikroprocesorjih lahko uporabnik s posebnim (t.i. *baznim*) registrom področje vektorjev prestavlja na poljubno mesto.

Priporočljivo je, da vsebine vektorjev izjem vedno določimo, tudi za izjeme, ki jih ne pričakujemo; če spregledamo možnost in se izjema vendarle zgodi, njen vektor pa ni določen, nadaljnje izvajanje ne bo več definirano. Za takšne izjeme običajno združimo strežne rutine v skupni diagnostični rutini. *Dolžina* vektorjev je določena z velikostjo naslovov, pri sodobnih mikroprocesorjih običajno 32 bitov oz. 4 zlogi. Vektorje običajno podajamo po njihovih *številkah*. Govorimo o t.i. številki vektorja izjeme ali kar številki izjeme.

Na pomnilniškem področju, kjer so vektorji izjem, je v končnih aplikacijah običajno bralni pomnilnik (ROM), v katerega fiksno sprogramiramo naslove strežnih rutin. Zelo ugodno je, da v tem primeru skok na strežne podprograme – razen inicializacijskega – izvedemo posredno preko vsaj enega ukaza v RAMu: vektor za izjemo kaže na naslov v RAM-u, na katerem se nahaja brezpogojni skok na dejansko strežno rutino v ROM-u. Te brezpogojne skoke vpišemo v pomnilnik ob inicializaciji in jih lahko kasneje spremenimo tako, da kažejo na druge rutine. Tako dobimo *dinamične* vektorje. Ob inicializaciji v operacijskem sistemu se v te vektorje vpišejo naslovi sistemskih strežnih oziroma diagnostičnih rutin. Med delovanjem programa jih lahko poljubno spremenimo in s tem dosežemo, da se ob izjemi izvrši alternativna strežna rutina.

8.2 Zgledi izjem

- *Reset* To je izjema, ki se izvede ob vklopu mikroprocesorja (ali ob signalu na krmilni liniji RESET).
- *Napaka na vodilu* Ta izjema se sproži, če mikroprocesor naslovi neko neobstoječo oziroma okvarjeno pomnilniško lokacijo. Ker gre za zunanjo izjemo jo lahko zazna samo posebno logično vezje zunaj mikroprocesorja. V primeru napake to vezje nato preko posebne krmilne linije sproži izjemo.
- *Nedovoljen ukaz.* Vsaka binarna kombinacija operacijske kode ne predstavlja razpoznavnega ukaza mikroprocesorja. Če mirkoprocesor zazna takšnko kombinacijo lahko sproži izjemo. Vzrok za takšno napako je v napake v programu ali v okvari programskega pomnilnika.
- *Deljenje z nič, itd.* Ta izjema se sproži pri izvajanju aritmetičnih ukaziov z nedovoljeno vrednostjo operandov.

- *Pasti.* Pasti so posebna vrsta izjem, ki jih sproži programer z ukazom. Čeprav gre navadno operacijsko kodo, pa jo mikroprocesor obravnava na enak način kot ostale izjeme. Pasti se običajno uporabljajo za klice sistemskih funkcij, ali pa kot prekinitvene točke.

- *Sledenje.* Izjema sledenja nam omogoča izvajanje programa ukaz za ukazom. Uporablja se za odkrivanje napak v programu tako, da sledimo izvajanju programa in opazujemo vsebine registorv, pomnilnika, ipd. Deluje tako, da se po izvedbi vsakega ukaza mikroprocesorja sproži izjema. Strežni program za izjemo sledenja pa nam nato prikaže na zaslonu vsebino posameznih registrov ipd. Seveda je sledenje znotraj strežne rutine onemogočeno.

8.3 Splošni model obravnave izjem

V splošnem lahko obravnavo izjeme znotraj mikroprocesorja razdelimo v več korakov:

1. Mikroprocesor zazna, da je prišlo do izjeme in ugotovi njen izvor. Za zaznavo izjeme, ki so posledica dogajanja znotraj mikroprocesorja poskrbi krmilna enota. Zunanje izjeme se signalizirajo preko določenih krmilnih linij (priključkov), ki jih mikroprocesor pravtako prestreže in obdela.

2. Mikroprocesor shrani trenutno stanje programa (kontekst). Stanje programa je potrebno shraniti zato, da ga bomo lahko po zaključku strežne rutine obnovili. Samo tako lahko zagotovimo, da strežna rztina ne bo vplivala na prekinjen program. Stanje programa predstavlja trenutna vsebina registrov mikroprocesorja. Pri starejših (in manjših) mikroprocesorjih so se ob izjemi shranili vsi registri. Pri novejših mikroprocesorjih z velikim številom registrov bi to bilo časovno zamudno. Zato se pri njih shranijo samo najnujneši registri (običajno sta to programski števec in statusni register). Za ostale registre mora poskrbeti programer sam. Običajno naredi to tako, da na začetku strežne rutine shrani tiste registre, ki jih bo le-ta uporabljala, tik pred zaključkom strežbe pa njihovo stanje znova obnovi. Kot shramba za kontekst procesorja se običajno uporablja sklad obstajajo pa tudi druge rešitve.

3. Mikroprocesor preide v nadzorni (zaščiten) način delovanja. Ta korak se izvede pri večini sodobnih mikroprocesorjev. Nadzorni način delovanja omogoča strežni rutini popolni nadzor nad izvajanjem mikroprocesorja za razliko od uporabniškega načina delovanja, kjer so možnosti nadzora nekoliko omejene. Na ta način se zavarujemo, da bi slabo napisan uporabniški program nekontrolirano vplival na delovanje mikroprocesorja. Enostavnejši mikroprocesorji različnih načinov delovanja ne podpirajo.

4. Mikroprocesor določi naslov strežne rutine. Kot smo že opisali zgoraj je lahko naslov strežne rutine pri enostavnejših mikroprocesorjih določen že z njenim izvorom. V tem primeru je naslov strežne rutine konstanten. Velika večina sodobnih mikroprocesorjev pa deluje preko vektrojev izjem, tako da se ob proženju izjeme določi samo njena številka. Tudi določanje številke (vektorja) izjeme lahko poteka na dva načina. Po prvem načinu številko vektorja določi mikroprocesor sam in je vezana na izvor izjeme. Drugi način se uporablja pri prekinitvah. Uvedli so konstruktorji sodobnih mikroprocesorjev z namenom, da bi lahko le-ta postregel prekinitve iz velikega števila najrazličnejših vhodno/izhodnih naprav. V tem primeru mikroprocesor zazna, da je prišlo do prekinitve. Namesto, da bi sam določil številko vektorja izjeme o njej povpraša kar vnodno/izhodni vmesnik, ki je prekinitev sprožil (to je t.i. prekinitveni prevzemni cikel). Programer ob inicializaciji sistema v V/I vmesnik vpiše želeno številko vektorja in pripravi ustrezni vektor izjeme.

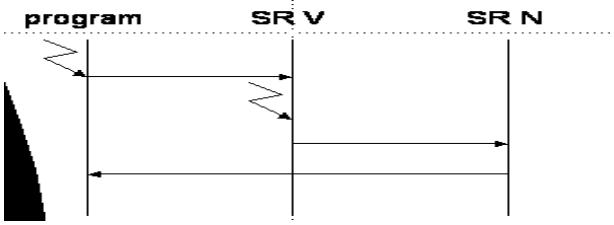
5. Začne se strežba izjeme. Ko je mikroprocesor določil naslov strežne rutine izjeme jo vpiše v programski števec in s tem prenese izvajanje programa na njo. Strežna rutina ustrezno poskrbi za vrok zaradi katere je izjema nastala. Strežna rutina je napisana v obliki podprograma, le da se zaključi z drugačnim (posebnim) ukazom.

6. Mikroprocesor obnovi stanje prekinjenega programa in nadaljuje njegovo izvajanje. V tem koraku se izvedejo aktivnosti, ki so obratne tistim v drugem koraku. Mikroprocesor obnovi stanje registrov, med drugim tudi vsebino programskega števca in prenese izvajanje na prvotni program.

8.3.1 Razreševanje sočasnih izjem in prekinitev

Pri normalnem poteku izvajanja programov mikroprocesorja se lahko zgodi, da se dve ali več izjem zgodi hkrati, oz. da se znotraj strežbe izjem lahko sproži nova izjema. Zato so izjeme organizirane hierarhično. Določene izjeme so bolj pomembne kot druge. Običajno so mikroprocesorji implementirani tako, da strežba pomembnejše izjeme onemogoči (zadrži) reakcijo na izjemo z isto ali manjšo pomebnostjo.

Posebni primer predstavlja sočasno proženje prekinitev. Ker vhodno/izhodne naprave lahko delujejo neodvisno druga od druge lahko dve ali več naprav proži prekinitve istočasno. Da ne bi strežba manj pomembnih naprav prekinjala strežbe bolj pomembnih, je so pri večini mikroprocesorjev posebej vpeljani posebni nivoji prioritet za prekinitve. Tako ima vsak izvor prekinitev tudi svojo prioriteto. Sprežbe prekinitev na nekem prioritetnem nivoju ne more prekiniti prekinitev na enakem ali nižjem prioritetnem nivoju. Mehanizem obravnave prioritet je rešen preko t.i. prekinitvene maske. Mikroprocesor si zapomnil do sedaj najvišjo prioriteto prekinitvene zahteve. Vse ostale zahteve po strežbi prekinitve z enako ali nižjo prioriteto pa so žamaskirane". Prekinitev z nižjo prioriteto bo postrežena po koncu strežbe prekinitve z višjo prioriteto.



Prekinitveno masko običajno predstavlja določeno število bitov v statusnem registru mikroprocesorja, ki hrani številko trenutne prioritete. Ob prekinitvi se (poleg ostalega) shrani tudi vrednost statusnega registra z maskirnimi biti, le-ti pa dobijo novo vrednost. Ob zaključku strežbe se v statusni register povrne prejšnje stanje in s tem dobijo priložnost tudi zahteve z nižjo prioriteto.

Masko lahko spreminja tudi programer sam. Če npr. želi, da prekinitve ne bi prekinjale nekega kriričnega dela programa, jo na začetku kritičnega območja poveča, na koncu pa ponovno zmanjša.

8.4 Programiranje strežnih rutin

Kot je bilo omenjeno že nekajkrat, strežne rutine za obravnavo izjem naj ne bi vplivale na obnašanje prekinjenega programa. Zato mora poskrbeti tudi programer. Na začetku mora programe shraniti vse tiste registre, ki jih bo strežna rutina uporabljala in se niso shranili že skozi normalni mehanizem za obravnavo izjeme.

Paziti je potrebno trudi na pomnilniške lokacije za začasne podatke. Le-te se ne smejo prekrivati s tistimi, ki jih uporablja osnovni program. Še najbolje je, če si za njih rezerviramo prostor na skladu.

Strežna ratuna se bi naj po možnosti zaključila v najkrajšem možnem času. Samo tako bo zakasnitev prekinjenega programa minimalna.

Strežna rutina mora odpraviti vrok izjeme. Če tega ne stori, se bo po zaključku strežbe izjeme le-ta ponovila. Pri prekinitvah, ki jih prožijo vhodno izhodne naprave zadostuje npr., da prečitamo prispeli podatke, včasih pa je potrebno ponastaviti določene statusne bite v vhodno/izhodnem vmesniku. V določenih primerih vzroka napake ne moremo odpraviti (npr. ob napaki na vodilu). V takem primeru mora programer zagotoviti, da so bo izvajanje programa zaključlo na kontroliran način.

Poglavje 9

Sodobne tehnike za povečanje zmogljivosti mikroprocesorjev

9.1 Pospeševanje ure

??

9.2 Matematični in drugi ko-procesorji

9.3 Pipeline

9.4 Branch prediction

9.5 Cache

9.6 Posebni ukazi

npr. za DSP

9.7 Multiprocesiranje

Poglavje 10

Podrobnejši pregled nekaterih sodobnih mikroprocesorjev in mikroračunalnikov

10.1 Arhitektura Pentiuma

cevovod, branch prediction, cache, MMX itd..

10.2 Arhitekturne značinosti PC računalnika

ISA, PCI, AGP, SCSI ..

10.3 Drugi sodobni mikroprocesorji

PowerPC, Alfa, MIPS, .. 64-bitni mikroračunalniki

Poglavje 11

Mikroprocesorji v vgrajenih sistemih

11.1 Sistemi v realnem času

11.2 Trdoživost vgrajenih sistemov