4. RISC arhitekture

4.1. Uvod in kratka zgodovina

Sama kratica RISC (reduced instruction set computers) ne pove dovolj o značilnostih arhitekture in organizacije takih procesorjev. Beseda reduciran-zmanjšan pove, da je število ukazov reducirano, kar pa očitno ni razpoznavna lastnost (n.pr. prvi mikroračunalniki so imeli majhen nabor ukazov, pa še daleč niso bili RISC). Podajmo zato raje spisek nekaterih lastnosti, ki bolj podrobno opredeljujejo arhitekturo:

1. Vsi ukazi naj se izvedejo v enem ciklu. Posledici te zahteve sta vsaj dve: ni možno uvesti kompleksnih ukazov, ker zahtevajo več izvršilnih ciklov in možno je skrajšati izvršilni cikel na čas, ki je potreben za izvršitev enostavnih podatkovnih transformacij (n.pr. aritmetične operacije prvega reda v stalni vejici).

2.S pomnilnikom delata le ukaza VLOŽI (LOAD) in SHRANI (STORE). Ta zahteva ima več različnih vplivov. Prvi je ta, da tako uvajamo arhitekturo tipa register-register. V nasprotju s prevladujočim prepričanjem v času, ko so se pojavile prve ideje o RISC-ih, omogoča to enostavno realizacijo kvalitetnih prevajalnikov. Odločitev vpliva tudi na preprostejšo interpretacijo ukazov in bolj čisto delitev rešitve problema zaustavljanja tekočega traku med aparaturno in programsko opremo. Ker moramo ceniti strokovnjake, ki plujejo proti toku, zapišimo, da je “neki” g. Seymour Cray že v šestdesetih letih (CDC6600) prisegal na arhitekturo register-register, stavil le na LOAD/STORE in višal procesno moč s tekočim trakom. Tem odločitvam je ostal zvest tudi v obdobju največje CISC ekspanzije.

3. Ožičeno krmiljenje. Ta zahteva izhaja pravzaprav že iz prve zahteve in je lahko tudi sporna. Če mikroprogramirano krmiljenje podaljša trajanje cikla, ostaja le še izbira ožičenega krmiljenja, ki je inherentno hitrejše. Nekaterim kompleksnim ukazom (predvsem operacije v pomični vejici se ni možno ogniti), zato novejše arhitekture vsaj za del množice ukazov in ALE, ki jih izvaja, dodeljujejo ponovno mikroprogramirano krmiljenje).

4.Majhno število ukazov in majhno število načinov naslavljanja. Ta zahteva omogoča enostavno interpretacijo, preprostejšo zasnovo prevajalnika in zmanjšan obseg operacijskega in posledično tudi krmilnega avtomata procesorja.

5. Stalen format ukazov. Ob opoštevanju te zahteve se predvsem poenostavi dekodiranje ukaza. To omogoča združitev dostave in dekodiranja v en cikel ter preprostejše generiranje krmilnih signalov.

6.Optimizacija časa procesiranja v fazi prevajanja programa. Prevajalnik pozna vnaprej potrebo po dostavi operandov, razpozna dele, ki so nespremenljivi in jih je možno izračunati vnaprej in podobno. Prevajalnik dela v bistvu tako, kot naj bi delal pisec mikroprogramov za WCS (Writable Control Store, R/W kontrolni mikroprogramski pomnilnik). Zato tudi nekateri avtorji govorijo o prevedenem programu-milikodi.

V razvoj RISC procesorjev so se do sedaj vključevale številne uspešne izkušnje iz preteklega razvoja računalnikov. Številne knjige v obravnavi začetkov navajajo le dva projekta RISC I in II ter MIPS. Vodilna raziskovalca na teh projektih in danes nesporni avtoriteti med arhitekti računalniških sistemov D.A. Patterson in J.L. Hennessy pa sta očitno (prvi to celo direktno priznava v publiciranem članku) nekaj vedela o govoricah povezanih s projektom 801, ki ga je v letu 1975 začel IBM. Projekt se je leta 1979 končal z delujočim prototipom na osnovi standardnih ECL vezij (takt 66 nsek ali 16 MHz!), z zelo učinkovitim naborom prevajalnikov (PL.8, Pascal) in bunkerjem. Zakaj so se v IBM odločili, da ne gredo v proizvodnjo v popolnosti verjetno ne bo nikoli znano. Znano pa je dejstvo, da so preprečili kakršnokoli publiciranje do leta 1982! Omenimo še ime vodilnega arhitekta, ki je vložil mnogosvežih idej v zasnovo procesorja prilagojenega za delo s teksti in celimi števili - George Radin.

Če želimo razumeti prodor nove arhitekture, moramo ponoviti predpostavke, ki so vodile v pospešen razvoj CISC arhitektur. Stroški razvoja programske opreme so naraščali, stroški produkcije in razvoja aparaturne opreme so padali. Izhod v boju proti naraščanju stroškov za programsko opremo so iskali v smeri razvoja programskih jezikov, ki so v večji meri izpolnjevali uporabnikove zahteve in nudili številne učinkovite konstrukte. Posledica tega trenda je bila, da se je semantična vrzel, to je vrzel med operacijami, ki jih ponujajo visoki programski jeziki (VPJ) in med operacijami, ki jih zagotavlja arhitektura, neprestano večala. Posledica te vrzeli so bile: neučinkovito izvrševanje, obsežni programi v strojni kodi in kompleksnost prevajalnika. Snovalci so skušali to vrzel premostiti z veliko množico ukazov (po pravilu kompleksnimi), velikim številom naslovnih načinov in poskusi realizacije stavkov VPJ direktno v aparaturni opremi. Vse te trende je najbolje zadovoljevala mikroprogramska krmilna shema. V okviru teh rešitev se je pojavila tudi ideja o DEL (Direct Executable Language, visokoprogramski jeziki za neposredno izvajanje).

CISC arhitektura naj bi torej:

-olajšala delo piscem prevajalnikov,
-izboljšala učinkovitost izvrševanja in
-zagotavljala podporo za še bolj kompleksne in sofisticirane VPJ.

Če sledimo globoki resnici, ki je vsebovana v pregovoru, da je pot v pekel tlakovana z dobrimi namerami, že vemo, kaj se je dogodilo. Snovalci so zapirali semantično vrzel in odpirali performansno vrzel. Na ta pojav so posredno opozarjale številne analitične študije izvajanja programov, ki so analizirale relativne frekvence ukazov, določenih zaporedij ukazov, spreminjanje globine zank, število in tipe operandov procedur, skratka vse, kar je J.C. Browne zajel pod pojmom model računanja[1].

Pot je bila še bolj brezizhodna, ker očitno snovalci (danes je to jasno, iz publikacij obravnavanega obdobja pa sledi, da ni bilo) niso upoštevali hitro se spreminjajočega razmerja med časom dostopa do hierarhično najvišjega pomnilnika (ali tudi povprečnim časom dostopa do pomnilnika) in časom trajanja izvršilnega cikla ukaza. Iz časov feritnih pomnilnikov izvirajoče razmerje več kot 10:1 se je v začetku uvajanja polprevodniških pomnilnikov še ohranilo, potem pa začelo pospešeno padati.

Če pa je dostop do pomnilnika dovolj hiter, ni več razlogov za združevanje operacij v kompleksno operacijo. Gledano s stališča hitrosti izvajanja ne pridobimo nič, saj hitrosti procesiranja ne ovirajo več potrebni dostopi do pomnilnika za dostavo ukaza.

[1] J.C.Browne:Understanding Execution Behavior of Software Systems, Computer, July 1984, str.83-87

Skoraj istočasno sta na univerzah Berkeley (1980 RISC I in 1983 RISC II) in Stanford (MIPS 1981) začela projekta zasnove procesorja v enem čipu, zasnovanega na popolnoma drugačnem pristopu, kot je prevladoval v takratnem snovanju mikroprocesorjev (n.pr Intel i432 in Motorola 68000). Ekipo na Berkeleyu je vodil D.A. Patterson. Imela je več izkušenj na področju snovanja aparaturne opreme, kar se kaže tudi v nekaterih odločitvah (koncept oken registrov). Ekipo na Stanfordu je vodil J.L.Hennessy in so, kot že ime projekta pove MIPS (Microprocessor without Interlocked Pipelined Stages), izhajali predvsem iz izkušenj pri pisanju prevajalnikov. V niz strojnih ukazov vstavijo NOP povsod, kjer bi prišlo do zaustavitve tekočega traku. Prevajalnik skuša NOP nadomestiti s strojnim ukazom, ki ne vpliva na pravilno izvedbo programa. Aparaturna oprema ne potrebuje gradnikov za detekcijo "mehurčkov" v cevovodu gradnikov za zaustavljanje tekočega traku.

4. 2. Analitične osnove za snovanje RISC arhitektur

4.2.1. Uvod

Že v uvodu smo omenili, da so nekatere publicirane analize o modelu računanja imele pomemben vpliv na razmišljanja o novi arhitekturi. Avtor pravi, da je model računanja definiran z:

- primitivnimi enotami računanja,
- pravili, ki preoblikujejo te primitivne enote računanja v strukture računanja,
- pravili za tvorbo naslovnega prostora, v katerem se izvajaj kompleksne strukture računanja,
- načini sinhronizacije paralelnega izvajanja in
- načini komuniciranja med strukturami računanja in njihovimi naslovnimi prostori.

Pojasnimo vsako od navedenih točk nekoliko podrobneje:

- **Primitivne enote računanja** definira množica ukazov procesorja. Določa operacije in podatkovne tipe, nad katerimi se lahko operacije izvajajo.

Najbolj običajen način **pretvorbe primitivnih enot** računanja v logične strukture računanja je uporaba niza ukazov, ki je običajno podatkovno odvisen.

- **Naslovni prostor** je množica informacij (programov in podatkov), ki je dostopna sestavljeni enoti računanja.
- **Sinhronizacija** je koncept, ki omogoča izvajanje ene same logične strukture računanja kot sestavo več nizov ukazov.
- **Komuniciranje** omogoča prenos informacij med naslovnimi prostori dostopnimi različnim nizom ukazov, ki lahko sestavljajo zaporedno ali pa vzporedno logično strukturo računanja.

Osnovno vprašanje pri zasnovi poljubne arhitekture je torej: katere operatorje in operande bomo izbrali in kakšne so njihove možnosti dostopa do pomnilnika v logičnih vzorcih.

Odločitev, da so operandi prisotni le v registrih, močno omejuje kompleksnost struktur podatkov. Prevajalnik mora poskrbeti, da s tem omejenim naborom realizira tudi bolj kompleksne strukture podatkov. Logična struktura računanja je lahko

popolnoma zakrita zaradi dostav in hranjenj, ki so potrebna zaradi realizacije kompleksnih struktur podatkov.

Posledica odločitve za določen tip arhitekture je, da so podatki zbrani s spremljanjem izvajanja programov, ki določajo pogostnost posameznih ukazov, značilnih skupin ukazov, načinov naslavljanj, uporabni brez rezerve le za izboljšanje te arhitekture.

Šele primerjava podatkov za različne arhitekture, naknadno izločanje specifičnih vplivov posameznih arhitektur in primerjave za različne vrste uporab nam lahko dajo podatke, ki so koristno vodilo pri definiranju arhitekture (n.pr. kateri ukaz, več prostora za niz registrov ali predpomnilnik) in tudi organizacije (izvajanje katerega ukaza je potrebno še optimizirati, kako dolg tekoč trak izbrati, vgraditi aparaturno ustavljanje tekočega traku).

Algoritme je možno preslikati v grafe in podgrafe računanja. Vrh grafa je uporaba primitivnega ukaza nad osnovno podatkovno strukturo, povezave med vrhovi pa zajemajo krmiljenje. Analiza takih grafov bo šele dala kvaliteten odgovor katere nove ukaze in katere tipe podatkov vpeljati.

Iz številnih analiz, ki so bile publicirane, je možno izločiti nekaj dejstev, ki imajo odločilno vlogo pri odločitvi za uvedbo koncepta RISC arhitekture.

Povzemamo jih po članku avtorja, ki je odločilno prispeval k zasnovi RISC I[2]. Avtor izhaja iz predpostavke, da je potrebno za kvalitetno odločitev o arhitekturi poznati naslednje tri skupine:

- Uporabljeni operandi (tipi, velikost, struktura in način njihove uporabe določajo organizacijo pomnilnika za njihovo pomnjenje in dostop).
  tipi operandov: cela števila, znaki, kazalci,...
  struktura operanda: skalar, polje, niz,...
  deklaracija operanda: globalen, argument procedure, lokalna spremenljivka,
  število operandov velikost in frekvenca dostopa za zgornje kategorije,
  obseg in narava lokalnosti, po možnosti določena individualno za vsako zgornjo kategorijo.
- Operacije, ki se izvajajo (določajo enote potrebne za izvajanje operacij in njihovo povezavo s pomnilnikom, pomembna je tudi relativna frekvenca operacij)
  testiraj, primerjaj, prištej, ...
  tip operacije (celoštevilčna, nad nizom,...)
  visokonivojske operacije (vhod/izhod, prikaži, ...)
- Zaporedje izvrševanja (določa krmiljenje med ukazi in organizacijo tekočega traku)
  prenos krmiljenja: pogojni/brezpogojni skoki in vejitve, klici, vračanja (frekvenca, razdalja, pogoji, napovedljivost, zgodnji izračun pogoja,...)
  količina in narava paralelizma, ki jo je možno izločiti (opomba: že v naravi von Neumannovih arhitektur je interes za nizkonivojsko paralelnost).

Poleg velikega obsega kvantitativnih meritev je potrebno prav tako preučiti korelacije med meritvami kar šele vodi k kvalitativnemu razumevanju.

Statične meritve (pregledovanje izvorne kode) dajo le malo koristnih informacij. Najbolj kvalitetne so tiste dinamične meritve, ki omogočajo iz izmerjenih parametrov za strojne ukaze (objektno kodo) izračun določenih parametrov tudi za izvorno kodo. Žal to ni vedno možno.

[2]M.Katevenis, Reduced Instruction Set Computers Architecture for VLSI, MIT Press 1985

Poleg direktne grobe metode masivnega merjenja eksistira še pristop, ki kombinira kvalitetno analizo izvorne kode, ki naj izloči tiste dele, v katerih se izvajanje programa najbolj pogosto zadržuje. Te dele imenujemo "kritične zanke". Kritične zanke lahko odkrijemo tako, da spremljamo kakšne *stroške izvajanja* je računalnik povzročil v določenem delu kode programa. Strošek je lahko:

- *porabljen čas*
- *število vrstic izvorne kode, ki so se izvršile,*
- *število dostopov do pomnilnika.*

V času snovanj prvih RISC arhitektur so si snovalci morali pomagati s sorazmerno grobimi orodji (programski monitorji), danes pa je možno ob uporabi aparaturnih monitorjev in logičnih analizatorjev dobiti kvalitetne podatke tudi o majhnih delih izvorne kode (n.pr. kratka zanka).

Povzemamo nekaj najbolj zanimivih podatkov v skupkih, ki so jih prispevali različni avtorji.

4.2.2. Meritve frekvence operacij

a) premiki register-pomnilnik 40%, vejitve 30%, aritmetika v stalni vejici 12%

b) VLOŽI 33%, SHRANI 10%, VEJI 14%, PRIMERJAJ 6%

4.2.3. Meritve frekvenc stavkov visokih programskih jezikov (VPJ)

a) prireditev 42%, if 36%, klic/vračanje 14%, zanke (for, while, until) 4%.

Če upoštevamo število potrebni strojnih ukazov za vsako kategorijo, dobimo:

b) zanke 37%, klic/vračanje 32%, if 16%. prireditev 13%.

Če upoštevamo število dostopov do pomnilnika, dobimo:

c) Klic/vračanje 45%, zanke 30%, prireditev 15%, if 10%

Vsi navedeni procenti so v originalnem tekstu podani z intervalom, ki ga pogojuje vrsta programov, ki so jih merili.

Vidimo da je operacija klic/vračanje pomembna, kakorkoli opazujemo računanje. Zato o njej še nekaj podatkov:

d) čas potreben za rokovanje s klicom procedure (hranjenje okolja) kot procent celotnega časa izvrševanja- 25%.

Zanimivi so tudi podatki o dolžini skoka pri vejitvenem ali skočnem ukazu, merjeni dinamično:

e) < 128 bajtov! 55% (za relativno naslavljanje zadošča 8 bitov)
še več < 16K bajtov 93% (še vedno le 14 bitov).

Tudi predstave o kompleksnosti prireditev so lahko varljive:

f) eno-členi izrazi 66%
dvo-členi izrazi 20%.

To se pozna tudi v
g) povprečnem številu operatorjev v izrazih: -0.76.

Zelo pogosto se torej izvajajo prireditve tipa:

A:=B,

ki jih lahko organizacija računalnika učinkovito podpre.
Zanimive so tudi meritve, kaj se dogaja z vsebino registrov, dokler so aktivni:

j) nad vsebino se ne izvrši aritmetična ali logična operacija 50%(20.5%-90%)
prišteje/odšteje se konstanta 25%
množi/deli se z konstanto 5%
izvede se operacija v pomični vejici 15%
uporabljena je kot indeks 40%.

Povzemimo na kratko: pomembni so klici in vračanja iz procedur, obvladati je potrebno vejitve, pomembna je enostavna aritmetika in naslavljanje. Zakaj? Izgubimo velik del časa, zanke in if so redki in izrazi so ponavadi zelo kratki, pol operandov se ne spremeni.

4.2.4. Meritve na operandih

a) celoštevilčne konstante 20%, skalarji 55%, polja/strukture 25%
sklicevanje na lokalne skalarje kot procent vseh sklicevanj >80%
sklicevanje na globalna polja/strukture kot procent vseh sklicevanj >90%

naslavljanje:

b) registrsko 32%, indeksiranje 17%, takojšnje 15%(konstante), relativno 11%
in vse ostalo 25%

Dolžina aktivizacijskega zapisa procedure:

	prevajalnik, urejevalnik teksta	kratki nenumerični programi
>3 argumenti	<7%	<5%
>5 argumentov	<3%	0%
>8 besed za argumente in lokalne spremenljivke	1%-20%	0%-6%
>12 besed za argumente in lokalne spremenlivke	1%-6%	0%-3%
<4	8%-15%	0%-2.5%
>8	1%-3%	0%-0.2%

Pomebna je ugotovitev, da je število argumentov v večini primerov med 3 in 5 in, da je tudi potrebno število besed za celoten aktivizacijski zapis v večini primerov manjše od 12.

Zanimivo bi bilo v povezavi z zgornjim še vedeti, kako se spreminja globina gnezdenja procedur v nekem smiselnem časovnem intervalu:

Avtorji te analize so poleg korektnega zaključka, da se globina gnezdenja sorazmerno malo spreminja, potegnili še zaključek, da to govori v prid konceptu prekrivajočih se oken registrov. Bolj korekten je zaključek: če imamo del sklada za aktivizacijske zapise realiziranega kot aparaturno opremo procesorja, je smiselno vsebino odplavljati v pomnilnik in vračati iz pomnilnika le na nekaj sprememb kazalca na vrh sklada in ne ob vsaki spremembi.

4.2.5. Meritve na kritičnih zankah

Še vedno najbolj kvalitetne in obsežne študije na to temo je objavil D.E.Knuth[3]. Kaj je ugotovil:

67% je prireditev, ena tretjina od teh tipa A=B,
11% IF, 9% GOTO in 3% DO,
3% CALL in
3% RETURN (procedur brez vračanja ti programi očitno niso imeli !?),
25% časa se porabi za V/I formatiranje.

Avtor pregleda, ki smo ga omenili v začetku poglavja, je dodal temu še nekaj ugotovitev za numerično računanje:

- struktura, ki dominira, je polje,
- dostop do strukture je pretežno regularen, neodvisen od vrednosti elementov strukture,
- izračun indeksov polja preko aritmetičnih operacij je smiselno nadomestiti s kazalci,
- prevladujejo operacije v pomični vejici,
- prisoten je nizko in visokonivojski paralelizem.

Njegove ugotovitve za programe, ki se ukvarjajo z urejanjem besedil in CAD so manj pregledne in občeveljavne (tokrat je programski jezik C):

- visok procent vejitev v urejanju besedil kot posledica primerjanja z 0,
- dostop do večine operandov je posreden preko lokalnih kazalcev, ostali dostopi so do lokalnih in globalni skalarjev,
- arhitekture tipa pomnilnik-pomnilnik zapravijo ogromno časa za dostop do celotne besede, čeprav potrebujemo le bajt,
- v seznamih so najbolj pogosti dostopi po principu predhodnik ali naslednik.

Dodajmo vsem ugotovitvam še spoznanje, da je prisoten nizkonivojski paralelizem, najbolj pogost primer tega paralelizma pa je izračunavanje naslova in izračunavanje rezultata (podatka).

4. 3. Skupne značilnosti RISC arhitektur

[3]D.E.Knuth, An empirical study of FORTRAN programs, Software-Practice and Experience 1, 1971, str.105-133

4.3.1 Primerjava pristopa pomnilnik-pomnilnik in register-register

V nasprotju s pričakovanji, da bodo prevajalniki zaradi možnosti uporabe kompleksnih ukazov generirali po pravilu krajše programe, so številni analitiki odkrili ravno obratne značilnosti. Zapisi programov v simbolični obliki so res krajši, število potrebnih binarnih mest pa ne bistveno manjše. Oglejmo si na dveh preprostih izsekih iz programov, kaj se dogaja ob izvrševanju izraza:

a:= b + c

pomnilnik-pomnilnik:

8	16	16	16
Add	b	c	a

I=56, D=96, M=152

register-register:

8	4	16		
Load	rB	b		
Load	rC	c		
add	rA	rB	rC	
Store	rA	a		

I=104, D=96, M=200

in zaporedja izrazov:

a:= b + c
b:= a + c
d:= d - b

pomnilnik-pomnilnik:

8	16	16	16
Add	b	c	a
Add	a	c	b
Sub	b	d	d

I= 168: D= 288, M= 456
V le-tem primeru nimamo kaj optimizirati!

register-register:

8	4	4	4
Add	RA	rB	rC
Add	RB	rA	rC
Sub	RD	rD	rB

I= 60, D= 0, M= 60

I- število bitov za dostavljene ukaze; D- število bitov za dostavljene operande;
M- število bitov dostavljenih in odloženih v pomnilnik.

V zadnjem primeru moramo zraven šteti še stavke za Vnos (LOAD) in Shranjevanje (STORE), ki pa jih lahko optimiziramo.

Čeprav je drugi zgled nekoliko privlečen za lase, vendar jasno kaže, kdaj imajo arhitekture register-register v izvajanju pa tudi v sami velikosti programa prednost pred arhitekturami pomnilnik-pomnilnik. Če so izvorni podatki maloštevilni in se nad njimi izvede več preslikav, potem so arhitekture register-register nedvomno favoriti. Pri katerih aplikacijah pa lahko še sploh računamo, da se nad izvornimi podatki izvede malo preslikav? To so morda le še skrajno enostavne aplikacije v računovodstvu, krmiljenja nekaterih procesov, kaj več pa bi še težko našli. Zgledov za obratno situacijo pa je vse polno.

4.3.2 Povzetek skupnih značilnosti RISC arhitektur

Avtor[4] povzema naslednje skupne značilnosti RISC arhitektur:

- en ukaz na cikel
- operacije register-register
- enostavni načini naslavljanja
- enostaven format ukazov.

Prva značilnost se v razširjenem zapisu glasi: en strojni ukaz na en strojni cikel. Strojni cikel je definiran kot čas, ki je potreben za dostavo dveh operandov iz registrov, izračun aritmetične operacije in hranjenje rezultata v register. V RISC arhitekturah strojni ukazi torej naj ne bi bili bolj kompleksni in se tudi naj ne bi dalje izvajali kot mikroukazi. Za takšne ukaze ne potrebujemo mikroprogramskega krmiljenja. Izvajajo se torej lahko hitreje kot njim enaki ukazi v mikroprogramsko krmiljenih procesorjih.

Druga značilnost se zrcali v dejstvu da imamo za dostop do pomnilnika le ukaza tipa VLOŽI (LOAD) in SHRANI (STORE). Odločitev ima posledice tudi na zmanjšano število ukazov v splošnem. Zahteva pa premišljeno dodeljevanje in uporabo registrov, kar je bil dolgo razlog, da so se takim arhitekturam ogibali. Še pred pojavom komercialnih izvedb so napisali prevajalnike, ki izkoriščajo le del množice ukazov (registerskega tipa predvsem) in dosegli do 10% hitrejše izvrševanje programov (IBM 370/168). Na osnovi analize podobnih zgledov kot je bil naš prvi, pa so obratno nekateri avtorji še konec sedemdesetih let predlagali arhiekture brez registrov.

Sam pojem enostavni načini naslavljanja je ohlapen. Dopuščeno je seveda takojšnje in neposredno naslavljanje, posredno pa le preko vsebine registra in ne preko vsebine pomnilniške lokacije. Cena, ki jo plačamo je majhna, saj je le redko potrebna večkratna posrednost. Zanimiva in pogosto izkoriščena možnost pa je, da poleg relativnega naslavljanja na osnovi PC uvedemo tudi relativno naslavljanje na osnovi vsebine kateregakoli registra.

Enostaven format ukazov pomeni posredno, da se moramo odločiti le za en ali največ nekaj formatov ukazov. Enostaven tudi pomeni, da je dolžina ukazov enaka! Polje za operacijsko kodo je fiksirano in pri dekodiranju ukazov ne sodelujejo druga polja, kaj šele polja, ki bi jih šele dostavili. Posledica toge delitve v polja je, da lahko dekodiranje ukazov in dostop do operandov v registrih začne istočasno. Enoten format ukazov pa ima tudi ugodno postransko posledico: ukaz nikoli ne sega čez mejo strani (ena beseda na ukaz!).

Vse te odločitve pa imajo ugodne posledice za doseganje zmogljivosti (performans) in možnosti implementacije v VLSI vezjih. Prvi faktor pri uporabi VLSI vezij je, da so zamuditve signala znotraj čipa bistveno krajše kot zamuditve med čipi. Posledica tega dejstva je, da je potrebno "dragoceno" površino silicija izkoristiti za

[4]Willliam Stallings, Computer Organisation and Architecture, Macmillan Publishing Company, 1986

tiste aktivnosti, ki nastopajo najbolj pogosto. To pa so prav gotovo enostavni ukazi in dostop do skalarjev. Drugi faktor je čas, potreben za zasnovo procesorja. Procesor zasnovan na reduciranem naboru ukazov, fiksnem formatu ukazov in omejenem naboru možnih načinov naslavljanja je možno proizvesti v verziji brez napak v bistveno krajšem času kot nekaj čipov ali tudi en sam čip, potreben za CISC arhitekturo. Ker je časovno okno določene polprevodniške tehnologije le od 3 do 5 let, igra trajanje razvojne faze produkta pomembno vlogo pri rentabilnosti celotnega projekta.

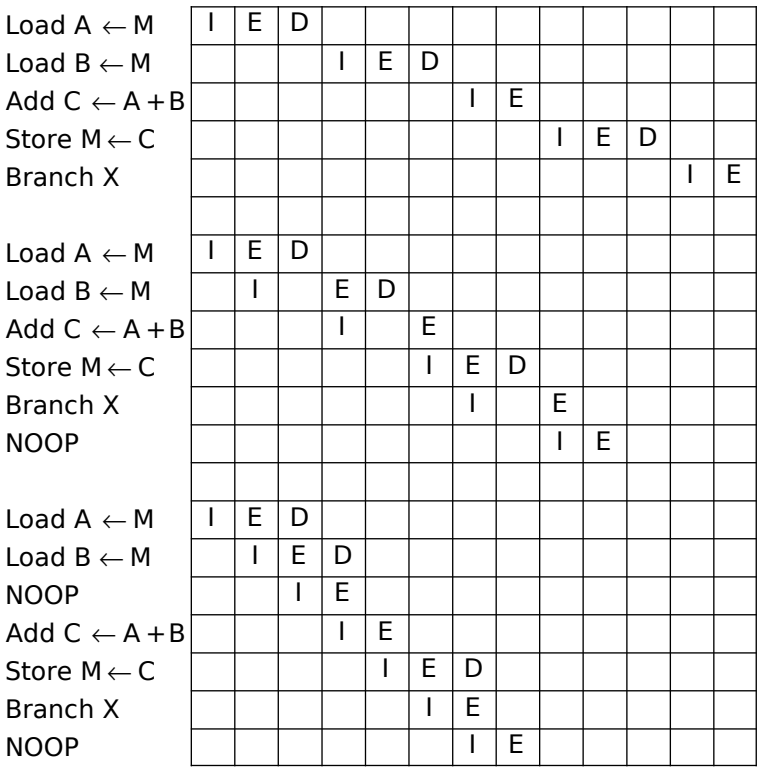
4.3.3 Tekoči trak (cevovod)

Ker je večina ukazov tipa register-register, je možno zasnovati tekoči trak, ki se po dejanski zmogljivosti dobro približa teoretični maksimalni zmogljivosti. Oglejmo si možne zasnove tekočih trakov.
Najbolj elementarna zasnova je trostopenjski tekoči trak, ki sloni na treh fazah:

- I-faza; dostava ukaza,
- E-faza; izvajanje aritmetično logične operacije z registri kot vhodi in izhodi ali izračun naslova v pomnilniku,
- D-faza; dostava pomnilnika v register ali obratno.

D-faza je prisotna le v redkih ukazih, zato se odločimo za tekoči trak z dvojnim prekrivanjem (I-faza in E-faza). V arhitekturah, ki omogočajo v eni fazi dostop do pomnilnika za ukaze in do pomnilnika za podatke hkrati, se lahko odločimo tudi za tekoči trak s trojnim prekrivanjem (I-faza, E-faza in D-faza). Slika 4.3.1 prikazuje razmere za namišljen izsek iz programa.

Iz S. 4.3.1. je razvidno, da tekoči trak zaustavljajo ukazi tipa Load in Store, vejitvene ukaze in ukaze, ki potrebujejo kot operand rezultat predhodnega ukaza. Vsi trije problemi so rešljivi vsaj v večini primerov in bomo njih reševanje obdelali v nadaljevanju.



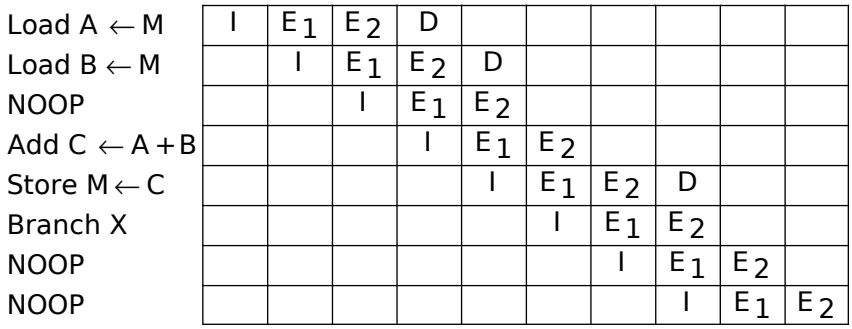
Slika 4.3.1. Tekoči trak zaustavljajo ukazi tipa Load in Store.

V dosedanjem prikazu smo molče privzeli dejstvo, da je čas potreben za posamezne faze enak. Ta predpostavka pa ne velja. Ker E-faza vključuje izvajanje operacije z dostavo operandov iz registrov in odlaganje rezultata v register, je dejansko daljša od I-faze, ki vključuje le dostavo ukaza (to velja še posebej, če se ukaz dostavlja iz predpomnilnika). Smiselno je razbiti E-fazo na dve podfazi E_1 in E_2:

E_1-podfaza; branje operandov iz registrov,

E_2-podfaza; izvajanje operacije in vpis v register.

Slika 4.3.2 prikazuje potek izvajanja programa, ki smo ga že obravnavali. Tekoči trak s četvernim prekrivanjem zahteva še več ukazov tipa NOOP (no operation, tudi NOP).



Slika 4.3.2. Tekoči trak s četvernim prekrivanjem zahteva še več ukazov NOOP.

4.3.4 Optimizacija tekočega traku

Operacije Load in Store, vejitve (Jump in Branch) in podatkovne odvisnosti naslednjega ukaza od predhodnega povzročajo zaustavljanje tekočega traku in zahtevajo aparaturno opremo, ki tako zaustavljanje omogoča. Če se mu želimo ogniti vstavimo NOOP. Za podatkovne odvisnosti, Load/Store in brezpogojne skoke in vejitve je s tem problem zaustavljanja rešen, vendar na škodo zmanjšanja zmogljivosti tekočega traku. Oglejmo si tehnike, ki omogočajo zamenjavo NOOP s smiselnimi ukazi in tako ohranjajo zmogljivost tekočega traku.

Zakasnjena vejitev: Do vejitve ne pride v samem ukazu ampak šele po izvršitvi ukaza, ki ji sledi. V prvem koraku ji mora slediti torej NOOP, ki pa jo skušamo nadomestiti z

a) ukazom, ki je pred vejitvenim ukazom, ne vpliva na izračun pogoja za vejitev in tudi ne spreminja vrednosti v registru, ki daje pogoj za vejitev. Pri brezpogojnih vejitvah in skokih ni problemov, pogojne pa zahtevajo izbiro le ozkega nabora ukazov ali pa izvedbo ukaza v dveh variantah. V prvi, normalni, ukazi vplivajo na izračun pogojev in njihov vpis v ustrezni register (CCR), v drugi, posebni, pa se izračun ne izvrši in vpis v register tudi ne.

b) pri konstruktih tipa IF...THEN...ELSE lahko prevajalnik pogosto v obeh vejah najde skupni ukaz, ki ga postavi pred vejitveni ukaz. Seveda tudi za takim ukazom veljajo omejitve navedne pod a).

Po principu zakasnjene vejitve lahko rešimo zaustavljanje tekočega traku pri ukazih Load in Store. Posebna tehnika je validacija registra. Pri izvajanju ukaza Load se ciljni register invalidira in je dostop do njegove vsebine nemogoč tako dolgo, dokler ni končan vpis nove vsebine vanj. Podobno velja za register udeležen v ukazu Store, da je nemogoče sklicevanje na njegovo vsebino in vpis vanj, dokler ni končana operacija hranjenja vsebine.

Prevajalnik poskrbi, da s premikom ukaz Load in Store doseže neprekinjeno delovanje tekočega traku. Slika 4.3.3 prikazuje na izmišljenem primeru korake generiranja zakasnjene vejitve.

Naslov	Normalna vejitev	Zakasnjena vejitev	Optimizirana zakasnjena vejitev
100	LOAD X,A	LOAD X,A	LOAD X,A
101	ADD 1,A	ADD 1,A	JUMP 105
102	JUMP 105	JUMP 106	ADD 1,A
103	ADD A,B	NOOP	ADD A,B
104	SUB C,B	ADD A,B	SUB C,B
105	STORE A,Z	SUB C,B	STORE A,Z
106		STORE A,Z	

Slika 4.3.3. Generiranje zakasnjene vejitve.

Učinek zakasnjene vejitve je najbolje viden iz časovnega diagrama za tekoči trak s trojnim prepletanjem na Sliki 4.3.4.

LOAD X ,A	I	E	D				
ADD 1,A		I	E				
JUMP 105			I	E			
STORE A ,Z					I	E	D
LOAD X ,A	I	E	D				
ADD 1,A		I	E				
JUMP 106			I	E			
NOOP				I	E		
STORE A ,Z					I	E	D
LOAD X ,A	I	E	D				
JUMP 105		I	E				
ADD 1,A			I	E			
STORE A ,Z				I	E	D	

Slika 4.3.4. Zakasnjene vejitve za tekoči trak s trojnim prepletanjem.

Podatkovna odvisnost: RISC arhitekture poznajo dva načina reševanja tega problema:

a) aparaturni način zagotavlja, da se lahko rezultat izvršitve neke operacije v ukazu i dostavi direktno kot operand ukazu i+1 (dostava mimo registrov). Avtorji si niso edini, koliko taka rešitev podaljša trajanje same E-faze.

b) uporaba prevajalnika zagotavlja, da v večini primerov do take situacije s preureditvijo kode ne pride.

Objektivna presoja obeh rešitev je blizu ugotovitvi, da se v primeru a) faza podaljša za 10% in v primeru b) prevajalnik reši problem podatkovne odvisnosti dveh zaporednih ukazov v 90% vseh slučajev. To vodi v ugotovitev, da sta obe predloženi rešitvi dokaj ekvivalentni v izboljšanju zmogljivosti tekočega traku.

4.3.5 Niz registrov ali predpomnilnik

Omenili smo že, da je bistvena značilnost RISC arhitektur zasnova množice ukazov po načelu register-register. Ta odločitev sama po sebi zahteva v procesorju določeno število splošnih registrov. Vendar pa ni določeno, kakšno je potrebno število registrov. Dobro je, da jih je čim več, vendar samo število registrov ne reši vseh problemov.

Oglejmo si situacijo ob klicu procedure. Raziskave so pokazale, da je mnogo bolj pogosta situacija vztrajanja na neki globini gnezdenja (intervalu) kot pa spuščanje v vedno večjo globino gnezdenja in potem vračanje nazaj. V proceduri uporabljamo lokalne spremenljivke, parametre, ki jih je predala klicoča procedura in globalne spremenljivke. Ponujata se dva načina reševanja problema zamenjave okolja (context switching).

a) Aparaturna rešitev problema sloni na konceptu delitve niza registrov na del za globalne registre in del za lokalne registre, ki je še naprej deljen v okna. Najboljša je izvedba s prekrivajočimi se okni, ki omogoča enostavnen prenos parametrov in vračanje rezultata. Hiba fiksne dolžine okna, ki jo lahko edino podpremo enostavno z aparaturno opremo je, da so okna nepopolno izrabljena ali pa premajhna.

b) Programska rešitev sloni na optimalnem prirejanju registrov spremenljivkam in konstantam. Za sam problem prirejanja je bilo že dolgo znano, da se ga prevede na problem barvanja grafov. Za ta problem je znano, da je NP- poln.

Eksponencialno naraščanje časa, potrebnega za prevajanje z optimizacijo, je dolgo odvračalo snovalce prevajalnikov od zamisli, da bi skušali optimizirati uporabo registrov. Šele raziskave v okviru projekta 801 in kasneje MIPS so pokazale, da je možno najti algoritme, ki v večini primerov poiščejo optimalno alokacijo registrov v času, ki je proporcionalen številu potrebnih alokacij. Problemi nastanejo le, če z omejenim številom registrov s temi algoritmi ni možno rešiti problema alokacije.

Prevladujeta dve rešitvi:

- Ena je direktna in v takem primeru ne išče možne rešitve alokacije ampak reši problem z odplavljanjem v pomnilnik in kasnejšim vračanjem iz pomnilnika.
- Druga rešitev sloni na uporabi algoritmov, ki vključujejo različne hevristike in skušajo najti rešitev alokacije v času, ki je proporcionalen kvadratu potrebnih alokacij.

Napredek tehnologije je zgornjo dilemo pri alokaciji registrov rešil na eleganten način. Vedno večja gostota gradnikov je omogočila znotraj VLSI vezja procesorja tudi realizacijo dveh predpomnilnikov (podatkovnega in ukaznega). Prvi rešuje hkrati dva problema: v njem so vsi podatki (skalarji, lokalne in globalne spremenljivke,...), ki jih program v trenutni fazi izvajanja potrebuje dovolj pogosto in je tako rešen problem zamenjave okolja. Visoka hitrost delovanja predpomnilnikov v okviru procesorja (on chip cache) minimizira zamuditve zaradi odplavljanja in vračanja vsebine v registre in s tem potrebo po velikem številu registrov.

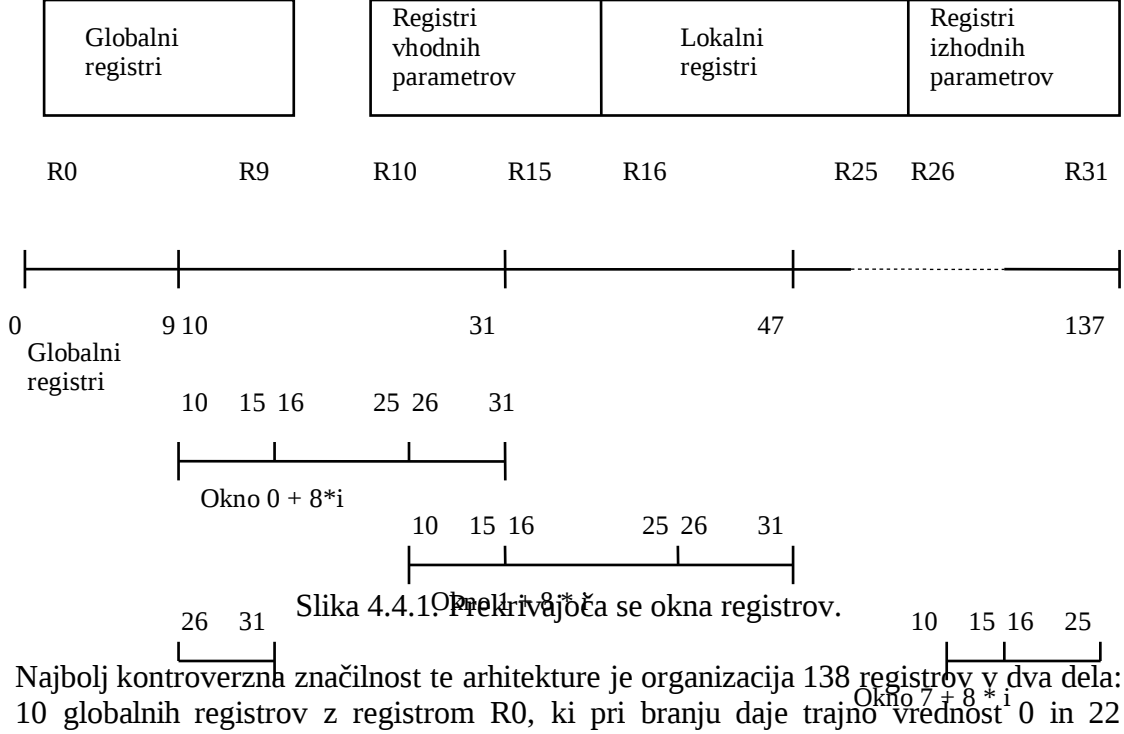
4.4. Zgledi posameznih arhitektur

Obravnavali bomo najprej tri prve prototipe kot ilustracijo začetnih razmišljanj o značilnostih RISC arhitekture. Tej obravnavi bomo dodali še obravnavo dveh komercialno zrelih aplikacij RISC arhitektur in tudi še nekaterih dodatnih zmožnosti.

Ne bomo obravnavali arhitektur firme MIPS, čeprav so bile do sedaj uporabljene v največ aplikacijah. Njihova glavna atraktivnost je v nizki ceni za zmogljivosti, ki jih prinašajo in bolj malo v širitvi konceptov RISC arhitektur.

Pri obravnavi prvih treh prototipov se bomo odločili za vrstni red, ki je običajen v popolnejših pregledih RISC arhitektur. Začeli bomo s predstavitvijo RISC I in II, ki je bil prvi publiciran, sledila mu bo obravnava MIPS in nato še obravnava 801.

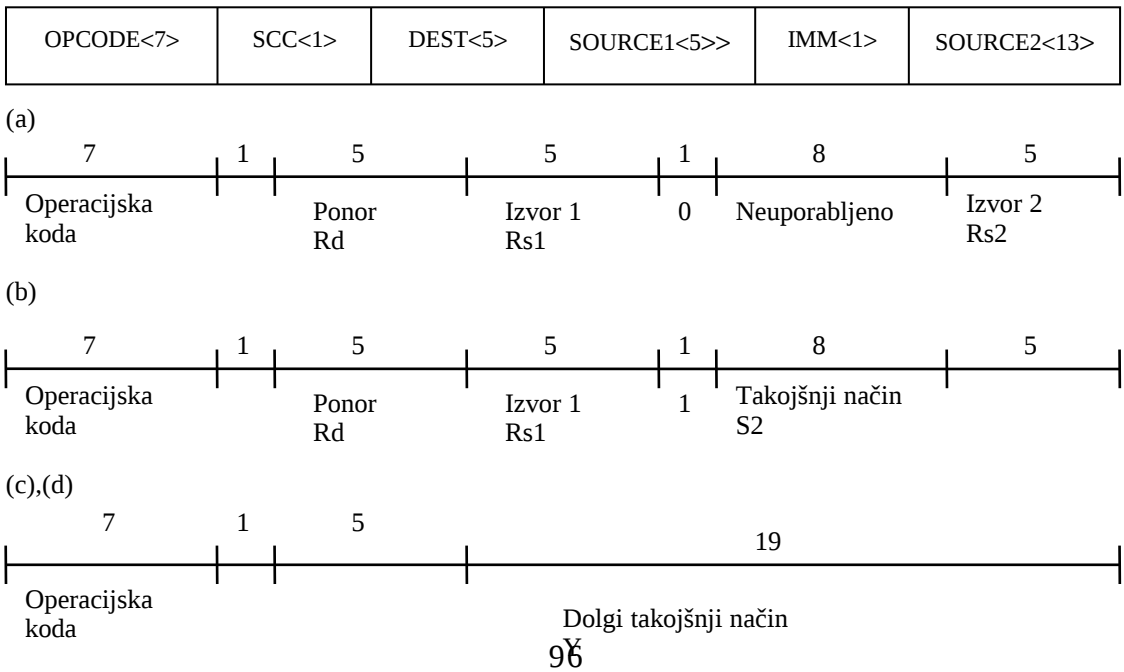
4.4.1. RISC I in II arhitektura



Slika 4.4.1: Prekrivajoča se okna registrov.

Najbolj kontroverzna značilnost te arhitekture je organizacija 138 registrov v dva dela: 10 globalnih registrov z registrom R0, ki pri branju daje trajno vrednost 0 in 22 prekrivajočih se oken trajno vrednost 0 in 22 prekrivajočih se oken. Vsako okno je razdeljeno v tri dele: dva dela po 6 registrov, ki služita kot vhodno/izhodni področji in 10 registrov za lokalno uporabo. Niz registrov je povezan v krožni pomnilnik tako, da se izhod zadnjega okna pokrije z vhodom prvega. Če pride do zasedbe vseh oken, se mora prvo okno umakniti v pomnilnik, od koder se ob sproščanju okna vrne vsebina zopet nazaj v prvo okno.

Arhitektura RISC II pozna 4 različne formate ukazov. Zanimivi sta dve polji dolžine 1: SCC in IMM (slika 4.4.1). Polje SCC določa ali se ob izvajanju ukaza izvede vpis v register pogojnih kod (koristna lastnost za implementacijo zakasnjenih vejitev). Polje IMM določa način določanja naslova drugega operanda. Ta je lahko v registru (uporabi se 5 najnižjih mest polja SOURCE2) ali pa se vsebina polja SOURCE2 tretira kot 12 bitna vrednost s predznakom. Format a) omogoča izbiro treh registrov, dveh za operanda in enega za rezultat operacije. Format b) omogoča izbiro registra za rezultat operacije, izbiro enega registra za vrednost enega operanda; drugi operand je ali 13 bitna vrednost ali odmik, ki se razširi levo do polne dolžine 32 bitov (zapis vrednosti je v dvojiškem komplementu). Formata c) in d) sta podobna. Polja SOURCE1, IMM in SOURCE2 so združena v eno in se uporabijo kot 19 bitni odmik pri vejitvenih ukazih (relativno na vsebino PC). Imamo 19 bitov v dvojiškem komplementu za odmik in različno uporabo polja za ponorni register (format c)) ali pogoj, ki ga testira vejitveni ukaz (format d))



Ponor
Rd ali pogoj
(4 LSB)

Slika 4.4.2. Format ukazov.

Arhitektura pozna 6 načinov naslavljanja. Morda so zanimivi le načini naslavljanja pomnilnika. Možno je direktno, indeksno in posredno naslavljanje. Arhitektura omogoča 8, 16 in 32 bitne celoštevilčne operacije vlaganja in hranjenja. Ukazi, ki delajo s podatki vedno uporabijo 32 bitne operande v registrih. Arhitektura ne pozna ukazov za operacije v pomični vejici. Realizirane morajo biti zato s pomikalnimi in seštevalnimi ukazi za stalen format in z uporabo vejitvenih ukazov nad štirimi možnimi pogoji Z, N, C in V. Ne pozna ukazov primerjaj in veji! Tabela 4.1 podaja načine izvedbe najbolj pogostih naslavljanj.

Tabela 4.4.1

		Vrednost		Vrednost operanda
Način naslavljanja	Format	Rs polje	S2/Y polje	
takojšnje	b	Rs	takojšnje	13 bitov, dvojiški komplement
registersko	a	Rs	Rs2	(Rs2)
absolutno	b	R0	takojšnje	M[13 bitni absolutni naslov]
z odmikom	b	Rs=R0	takojšnje	M[(Rs) + 13 bitni odmik]
relativno	c		takojšnje	M[(PC) + 19 bitni odmik]
registrsko posredno	b	Rs=R0	0	M[(Rs)]

Med RISC I in II je razlika v tehnologiji in z njim povezanim taktom ter razlika v dolžini tekočega traku. Medtem ko RISC I pozna le dve stopnji (dostava in izvajanje) ima RISC II tri stopnje: dostava, izvajanje in hranjenje rezultata. Dodatno pozna RISC II relativno naslavljanje (glede na vsebino PC) z 19 bitnim odmikom. Seznam ukazov RISC I je podan na sliki 4.4.3.

ukaz	Operandi	Rezultat	Komentar
ADD	Rs,S2,Rd	Rd←Rs+S2	prištej celo število
ADDC	Rs,S2,Rd	Rd←Rs+S2+prenos	prištej s prenosom
SUB	Rs,S2,Rd	Rd←Rs-S2	odštej celo število
SUBC	Rs,S2,Rd	Rd←Rs-S2-prenos	odštej s prenosom
SUBR	Rs,S2,Rd	Rd←S2-Rs	odštej celo število
SUBCR	Rs,S2,Rd	Rd←S2-Rs	odštej s prenosom
AND	Rs,S2,Rd	Rd←Rs&S2	logično in
OR	Rs,S2,Rd	Rd←RsvS2	logično ali
XOR	Rs,S2,Rd	Rd←Rs⊕S2	logično izključno ali
SLL	Rs,S2,Rd	Rd←Rs pomaknjen za S2	logični pomik levo
SRL	Rs,S2,Rd	Rd←Rs pomaknjen za S2	logični pomik desno
SRA	Rs,S2,Rd	Rd←Rs pomaknjen za S2	aritmetični pomik desno
ukaz	Operandi	Rezultat	Komentar

LDL	(Rx)S2,Rd	Rd←M[Rx + S2]	vloži dolgo
LDSU	(Rx)S2,Rd	Rd←M[Rx + S2]	vloži kratko nepredznač.
LDSS	(Rx)S2,Rd	Rd←M[Rx + S2]	vloži kratko predznačeno
LDBU	(Rx)S2,Rd	Rd←M[Rx + S2]	vloži bajt nepredznačen
LDBS	(Rx)S2,Rd	Rd←M[Rx + S2]	vloži bajt predznačen
STL	Rm,(Rx)S2	M[Rx + S2]←Rm	shrani dolgo
STS	Rm,(Rx)S2	M[Rx + S2]←Rm	shrani kratko
STB	Rm,(Rx)S2	M[Rx + S2]←Rm	shrani bajt
JMP	Po,(Rx)S2	pc←Rx + S2	pogojni skok
JMPR	Po,Y	pc←pc + Y	pogojni skok relativno
CALL	Rd,(Rx)S2	Rd←pc(naslednja) pc←Rx + S2 CWP←CWP - 1	klic procedure in sprememba okna
CALLR	Rd,Y	Rd←pc(naslednja) pc←pc + Y CWP←CWP - 1	klic procedure relativno spremeba okna
RET	(Rm)S2	pc←Rm + S2 CWP←CWP + 1	vrnitev spremeba okna
CALLINT	Rd	RD←pc CWP←CWP - 1	prepreči prekinitve spremeba okna
RETINT	(Rm)S2	pc←Rm + S2 CWP←CWP + 1	dovoli prekinitve spremeba okna
LDHI	Rd,Y	Rd<31:13>←Y Rd<12:0>←0	vloži takoj v višji del
GTLPC	Rd	Rd←pc	za odložen skok
GETPSW	Rd	Rd←PSW	vloži ststusno besedo
PUTPSW	Rm	PSW←Rm	postavi statusno besedo

Slika 4.4.3. Spisek ukazov RISC procesorja.

4.4.2 Mikroprocesorji MIPS

J.L.Hennessy je v preglednem članku[5] zapisal dve izhodišči, ki imata še danes vso težo:

"Izbira ukazov mora biti kompromis med primernostjo za kodiranje programov in performansami implementacije množice ukazov.
Konflikt, ki je inherenten zgornjemu kompromisu lahko najbolje reši obvladovanje kompleksnosti krmiljenja programa na nivoju programske opreme (prevajalnik)."

Predložil je tudi svoj termin za RISC arhitekturo. To naj bi bila po njegovem aerodinamična (streamlined) arhitektura.

Zasnova temelji na naslednjih predpostavkah:

- izbrana tehnologija (NMOS) favorizira uporabo velikega števila počasnih

[5]John L. Hennessy, VLSI Processor Architecture, Reduced Instruction Set Computers, IEEE Computer Society Press, 1986

gradnikov,

- komunikacija je dražja kot računanje,
- zamude v komunikacijah na čipu in med čipom in okolico se bistveno razlikujejo,
- VLSI arhitektura favorizira uporabo pomnilnika na procesorskem čipu.

Ta izhodišča so ob zahtevi po čimhitrejši aparaturni opremi (AO) preslikali v dve zahtevi:

1) minimizirati cikel ure v sistemu,
2) minimizirati število ciklov potrebnih za izvajanje vsakega ukaza

a) Upravljanje pomnilnika

S procesorji MIPS (Microprocessor without Interlocked Pipelined Stages) so želeli zagotoviti hitro preklapljanje med procesi. To so dosegli z vključitvijo identifikatorja procesa v virtualni naslov. Uporabili so lahko en sam linearen naslovni prostor. Uporabljena tehnologija ni omogočala realizacije celotne preslikave naslova v procesorskem čipu. Vsak proces ima v virtualnem naslovnem prostoru predpomnilnik (cache = kašča) in prostor za kodo in sklad, vmesni prostor, kamor se lahko širita, pa je odvisen od števila maskiranih MSB, ki se dinamično podrejajo procesu.

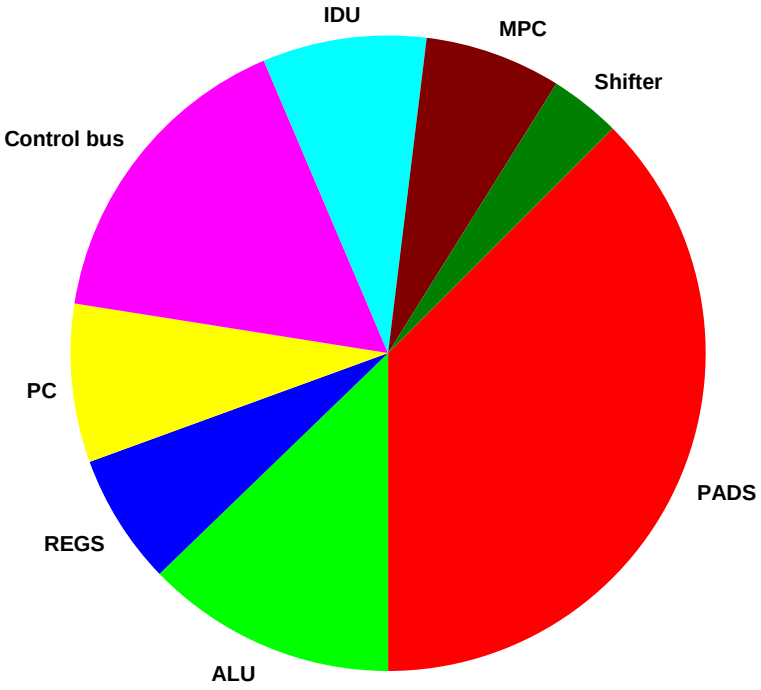
b) Množica ukazov

V izbiri ukazov je bilo vodilo pomik funkcij iz AO (aparaturne opreme) v PO (programsko opremo), tekoči trak je za uporabljeno tehnologijo maksimalno hiter in ima pet stopenj v dveh in pol ciklih. Cikel za dostavo in dekodiranje zajema dve stopnji, enako cikel za izvrševanje, zadnja stopnja je odlaganje rezultata v register. AO tekočega traku ne zagotavlja zaustavljanja, kadar je nek register ponor in izvor istočasno. PO mora poskrbeti za tak prevod, da se ognemo objemu. Zamuditvam v tekočem traku se ni možno ogniti, prevajalnik mora načrtovati minimalno primerno zakasnitev.

MIPS pozna samo naslavljanje besed. Dostop do bajtov omogočajo nekatere ukaze za delo s kazalci na bajte, ki so naslovi dolžine ene besede (najnižja bita določata pozicijo bajta znotraj besede).

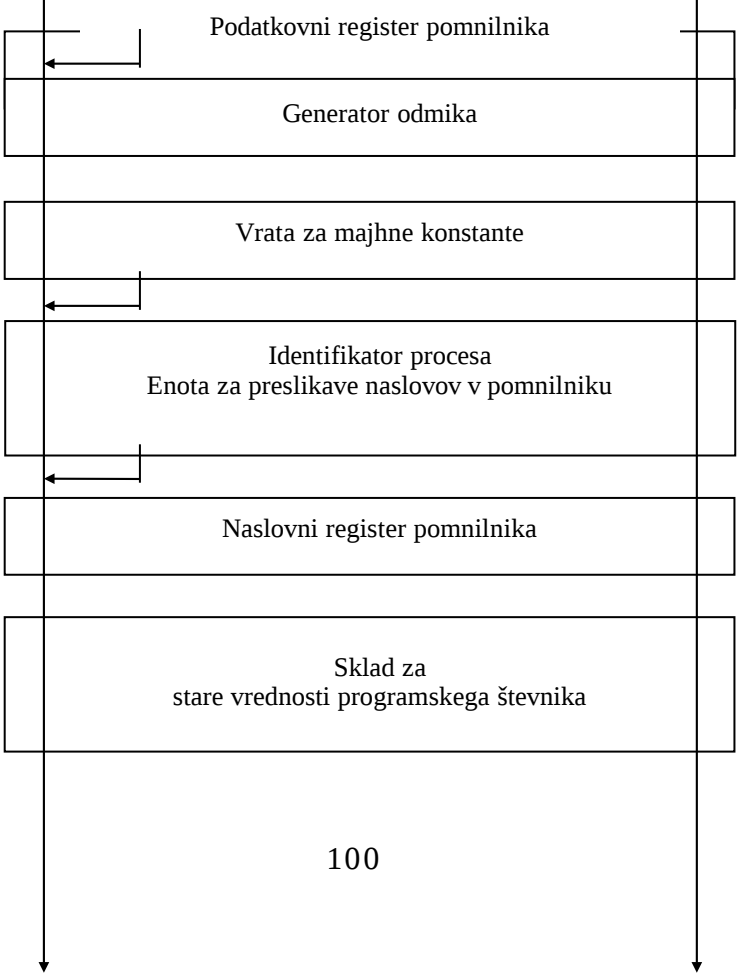
Procesor ima 16 ortogonalnih splošnih registrov. Ukazi delajo nad registri (2 ali 3 operandi). Podpora celoštevilčnemu množenju in deljenju sta posebna ukaza, ki zagotavljata dva bita iz Boothove množilne sekvence in en bit iz delilne sekvence.

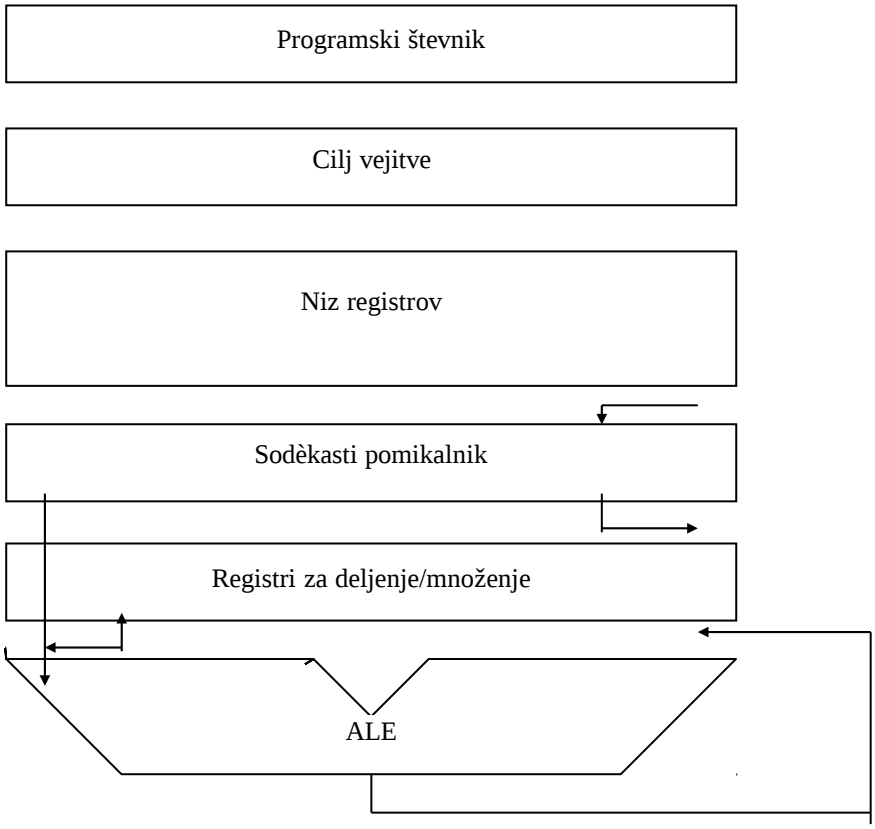
Prevajalnik bi želel imeti enostavno, dobro strukturirano množico ukazov. Ta zahteva pa se tepe s ciljem izpostaviti vse operacije v procesorju. Odločili so se za dva nivoja. Visoki programski jeziki se prevajajo v zbirnik, v katerem tudi lahko programirajo uporabniki in je lepo strukturiran. V ukazih ni vidno nič, kar bi lahko povzročilo trenje ali zatikanje tekočega traku (streamlined). Ti ukazi se v drugem nivoju prevedejo-reorganizirajo v zaporedje strojnih ukazov. Pri tvorbi tega zaporedja se upoštevajo posebnosti organizacije, kot je izvrševanje dveh AL ukazov v enem ciklu, izvrševanje VLAGALNE in AL ukaza v enem ciklu, preurejanje niza strojnih ukazov, da se ognemo objemom, pripravi operandov v registre, da bodo na voljo ob klicu procedure, reševanje odložene vejitve, razširitev makrojev.



Slika 4.4.4. porazdelitev porabe moči.

Procesor ne pozna registra pogojnih kod. Vse vejitvene ukazi so zato tipa PRIMERJAJ in VEJI. To rešitev omogoča delitev izvršilnega cikla na dve stopnji, v prvi se izvrši primerjava, v drugi izračuna naslov vejitve. Reorganizator (reorganiser) primerja možne prevode niza ukazov v zbirniku v niz elementarnih operacij-strojnih ukazov, ki se lahko deloma izvršijo tudi paralelno in izbere časovno najbolj ugodno možnost.





Slika 4.4.5. Podatkovne poti MIPS

Po navedbah avtorjev projekta naj bi kombinacija aerodinamične strukture cevovoda in optimizacija, ki jo da reorganizator zagotavljali faktor dve v izboljšanju performans procesorja.

Hitrost izvajanja operacij v procesorju je močno odvisna od zakasnitev signala na podatkovnih in krmilnih poteh. Te zakasnitve je možno skrajšati le ob pogoju, da se večina moči troši za operacije na teh poteh. Pomembne so tudi minimalne zakasnitve v izvedbi osnovnih aritmetičnih operacij, kar zagotavlja izvedba popolnega drevesa s prenosom vnaprej. Signala tvori in razširi sta generirana na osnovi dveh bitov sumandov.

Sliki 4.4.4 in 4.4.5 prikazujeta porazdelitev porabe moči in podatkovne poti MIPS.

4.4.3. RISC arhitektura IBM 801

V letu 1975 je skupina približno 20 raziskovalcev na IBM WRC pričela z raziskavami miniračunalnika, prevajalnika in krmilnega programa s ciljem doseči bistveno ugodnejše razmerje cena/zmogljivost za programe napisane v visokonivojskih programskih jezikih kot pri obstoječih sistemih. Ime je projekt dobil preprosto po številki poslopja, v katerem so tekle raziskave.

a) Osnovni koncepti.

Izvedba z enim ciklom je morda najbolj očitna značilnost arhitekture 801. Izvršilni cikel je en sam, precej primitiven in direkten (brez alternativ) strojni cikel.

Res je, da realizacija kompleksnih ukazov, ki zahteva več ciklov za izvrševanje z uporabo ožičene logike, vodi v bistveno hitrejše izvajanje v primeri z isto funkcijo programirano kot zaporedje primitivnih ukazov. Tej ugotovitvi ni moč oporekati.
Raziskave so pokazale, da primitivni ukazi zavzemajo pretežen del izvajanja programa. Frekvence kompleksnih ukazov so tako nizke, da ne morejo nadomestiti izgube zaradi podaljšanja izvajanja za en sam cikel (n.pr. za dostavo iz mikroprogramskega pomnilnika). Pri tem pa še nismo upoštevali zvišanja stroškov.

Pogosto so se tej dodatni opremi, ki je potrebna za realizacijo bogate množice kompleksnih ukazov, skušali ogniti s preprostim mikroprogramiranjem kompleksnih funkcij. To pa se ujema tudi s ciljem projekta 801.

Moč vertikalne mikrokode je le v tem, da počiva v zelo hitrem bralnem pomnilniku. To sili arhitekta, da vnaprej določi katere subrutine, makroji in ukazi bodo dobili mesto v mikroprogramskem pomnilniku. Če pa se odločimo za ukazni pred-pomnilnik, imamo še boljše razmere. Ob algoritmu LRU bomo imeli v njem ne samo makroje in zgoraj našteto ampak celotne sistemske procedure.

Programiranje kompleksnih funkcij kot programske opreme in ne kot mikrokode ima tri prednosti:

- procesor je lahko prekinjen na meji med mikroukazi brez posebnega dodatnega vložka v aparaturno opremo. Arhitekture s CISC morajo pri številnih ukazih prekinitev prepovedati med izvajanjem ukazov (n.pr. MOVE),
- prevajalnik lahko z optimizacijo najde dele kompleksnih funkcij, ki jih lahko izloči iz zank, razdeli ali združi z drugimi deli in
- dele kompleksnih ukazov je možno izračunati v času prevajanja ali pa za posebne primere ukazov (n.pr. množenje s konstanto) poiskati bolj optimalno zaporedje primitivnih ukazov.

Taka odločitev ima tudi pomanjkljivosti. Najbolj pomembna je vsekakor ta, da je lahko množica mikroukazov zasnovana tako, da njene operande direktno določajo polja v ukazih. V arhitekturi 801 mora prevajalnik poskrbeti za dostavo operandov v registre, če so potrebni.

Seveda pa je računalnik, katerega ukazi se izvršujejo učinkovito in hitro, atraktiven le, če skupno število ukazov, ki se morajo izvršiti za izvedbo naloge ne presega števila potrebnega pri CISC. Kasnejše meritve so pokazale, da je potrebno število za sistemske programe enako, pri komercialnih in znanstvenih aplikacijah pa lahko naraste do 50%.

b) Prekrivajoč se dostop do pomnilnika

Raziskave so pokazale, da bi pri 801 okoli 30% ukazov zahtevalo dostop do pomnilnika za podatke in da je še dodatno od 10% do 20% vejitvenih ukazov takšnih, da je potrebna vejitev. Če še upoštevamo, da so številne aplikacije tudi V/I zahtevne, nas ne preseneča dejstvo, da mora procesor često čakati več ciklov.

Drugi glavni cilj projekta je bil zato organizirati takšno pomnilniško hierarhijo in razviti tako sistemsko arhitekturo, ki bo minimizirala nedelavni čas procesorja zaradi dostopa do pomnilnika. Dostopni čas predpomnilnika (PP) mora biti konsistenten s strojnim ciklom procesorja. Izbrali so "odloži v PP" strategijo, tako da približno 10% potrebnih odlagalnih ukazov ne more resno zmanjšati zmožnosti.

Ker procesor potrebuje nov ukaz v vsakem ciklu in vrh vsega potrebuje podatke za/iz pomnilnika, se niso mogli odločiti za en sam skupen pred-pomnilnik. Ločili so ga v dva dela: podatkovni in ukazni in tako dejansko podvojili pasovno širino

in omogočili hkraten prenos podatkov in ukazov med primarnim pomnilnikom in predpomnilnikoma.

Odlaganje podatka lahko spremeni ukaz. Oba predpomnilnika bi morala biti sinhronizirana. To bi zahtevalo zelo kompleksno krmiljenje. Ker je vse od uvedbe indeksnih registrov realna potreba po spreminjanju ukazov v pomnilniku izredno upadla, so se arhitekti 801 odrekli tej možnosti in tudi niso predvideli opreme zanjo.
V običajnih sistemih mora V/I skozi predpomnilnik. Logično je to nujno, vendar je v cenenih sistemih to tudi fizično rešeno tako. Rezultat je takšen, da procesor čaka med V/I prenosom in po končanem prenosu vsebina predpomnilnika ne ustreza več delavni množici procesa, ki se izvaja.

Ker V/I upravljalnik ve za področje prenosa, obseg prenosa in čas prenosa, lahko PO pravilno sinhronizira predpomnilnika in lahko V/I aparaturna oprema prenaša direktno V/I primarnega pomnilnika. Tako tudi, če je polovica pasovne širine primarnega pomnilnika izkoriščena za V/I, zmogljivost procesorja na videz ni zmanjšana.

Vseskozi se ponavlja enaka strategija. Če je neka sistemska funkcija draga ali počasna v splošni obliki, vendar lahko PO prepozna njeno degenerirano obliko (ali lahko celotno funkcijo prenese iz časa izvajanja v čas prevajanja), se ta funkcija izloči iz AO v PO, kar rezultira v nižjih stroških in izboljšani zmogljivosti.

Dober primer te strategije je dostava v predpomnilnik. Vrstica predpomnilnika je dolga 32 bajtov, medtem ko je najdaljša naslovljiva vsebina v primarnem pomnilniku le 4 bajte. Če želimo dostavo v besedo, ki ni v predpomnilniku je potrebno dostaviti celotno vrstico (sledi lahko namreč dostava naslednje besede iz iste vrstice). Večinoma je dostava prva dostava v območje, ki je bilo na novo dodeljeno programu. AO ne ve, da ne bo več potrebovala stare vsebine v predpomnilniku, prevajalniku pa je to popolnoma jasno. Definirali so eksplicitne ukaze za rokovanje s predpomnilnkom, tako da PO lahko izloči nepotrebno vlaganje v in shranjevanje iz predpomnilnika.

Običajna PO privzema, da je pomnilnik naključno naslavljan. Vsak služnostni program v nadzorniku in podsistemih ima lasten začasni pomnilnik. V visokih programskih jezikih se ti programi kličejo enako kot subrutine in dobijo prostor za začasen pomnilnik na enotnem skladu, kar rezultira v večji uporabi vrstic predpomnilnika in tako v višjem procentu zadetkov.

Čeprav lahko za večino ukazov in podatkov, ki jih potrebuje procesor trdimo, da so v predpomnilniku, moramo pri konvencionalnem procesorju čakati, da je končana dostava operanda iz pomnilnika ali pa pri vejitvenem ukazu dostava naslednjega ukaza, če je pogoj izpolnjen.

Zahtevni procesorji često tvorijo "zgodovino" vejitev ali dostavljajo vnaprej obe veji. V 801 so se odločili za drugačno strategijo. Z majhnim številom aparaturnih primitivov lahko PO (prevajalnik) preuredi program tako, da ostane semantika nespremenjena, AO pa lahko ta prosti čas enostavno premosti z uporabnim delom.

Pri vlaganjih v register ta register zaklenemo. Zaklenjen ostane tako dolgo, dokler ni vanj vložena vsebina. Če ukazi, ki sledijo vlagalnem ukazu, ne uporabljajo zaklenjenega registra, izvrševanje ni prekinjeno. Prevajalnik lahko v večini primerov poišče take ukaze.

Pri vejitvenih ukazih so arhitekti 801 vpeljali za vsak ukaz še alternativno obliko VEJI Z IZVRŠEVANJEM. Po semantiki so identične vejitvenim s to razliko, da se med dostavo naslednjega ukaza izvaja ukaz, ki sledi vejitvenemu. Pogoj je le, da ta ukaz ne vpliva na vrednost pogoja in, da je originalno pred vejitvenim ukazom.

c) Sistem zasnovan na prevajalniku

Množica ukazov običajnih procesorjev je zasnovana na predpostavki, da bo mnogo programerjev programiralo v zbirniku. To je motiviralo definiranje kompleksnih ukazov skoraj tako močno kot razpoložljivost hitrih krmilnih pomnilnikov. Toda z vedno bolj prevladujočo uporabo visokih programskih jezikov in omejitvijo uporabe zbirnika le v primerih ostrih zahtev (časovnih ali prostorskih) je možno s prevajalnikom tvoriti objektno kodo, ki je dovolj blizu najboljši "ročni kodi". Za zelo specifične ukaze so na voljo učinkovite procedure.

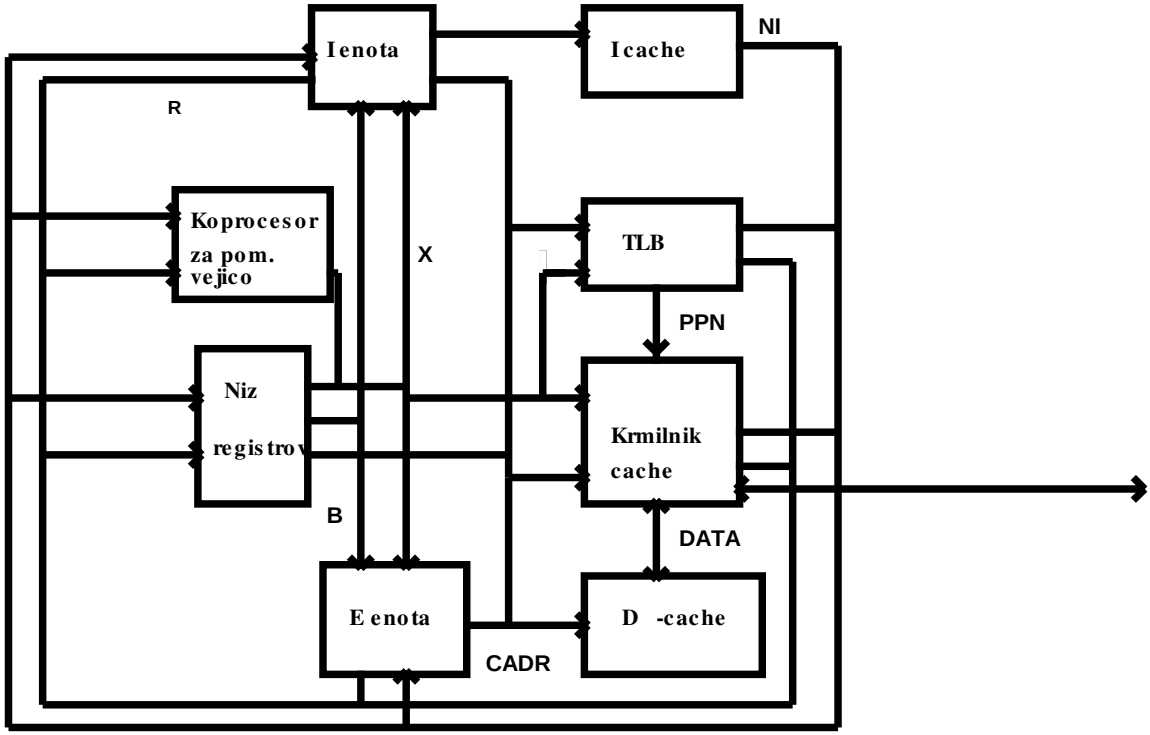
Rezultat vsega omenjenega je sistem, v katerem je le manj kot 1000 vrstic kode nadzornika in nekaj kompleksnih ukazov zapisanih v zbirniku. S tem je za organizacijo računalnika odpadla zahteva, da naj bo takšna, da omogoča enostavno programiranje v zbirniku. Le nekaj programerjev nadzornika in prevajalnika mora zbirnik res poznati in tako je bilo možno odločitve arhitektov 801 nasloniti le na potrebe teh nekaj programerjev in na tehtanje razmerja stroški/zmogljivost.

Arhitektura 801 je bila definirana kot množica operacij v času izvajanja, ki jih ni možno:

- premakniti v čas prevajanja,
- izvršiti bolj učinkovito z objektno kodo, ki jo tvori prevajalnik, ki razume visoko-nivojske namere programa ali
- bolj učinkovito implementirati v ožičeni logiki kot v ekvivalentnem zaporedju ukazov PO.

Predpostavka, da bodo programi prevedeni s prevajalnikom, omogoča tudi boljšo zaščito uporabnikov med sabo. Če predpostavimo, da so nekatere kritične pasaže prevajalnika korektne (niso zlonamerno spremenjene) in da so vsi programi, ki se izvajajo prevedeni, med njimi ne more prihajati do nedopustnih (namernih ali nenamernih) interakcij. To je zagotovljeno v času prevajanja in povezovanja in po potrebi z vstavljanjem ukazov pasti tudi še v času izvajanja.

Skladno s pravili lepega obnašanja je bil dan povdarek podpori strukturiranosti programa. To zahteva, da je za številne kratke subrutine in procedure povezovanje optimalno. Vse je storjeno, da so parametri ob klicu v registrih in se tako procedura izvede z enakim dodatnim bremenom kot vejitev.



Slika 4.4.6. Zasnova HP procesorja.

4.4.4 HP arhitektura

V letu 1982 je firma HP pričela z raziskavami na novi arhitekturi računalniških sistemov, ki jo je komercialno dovolj atraktivno poimenovala s HP (high precision). V letu 1984 je že lahko ponudila prve procesorje, v letu 1985 pa prve kompletne sisteme. Ti so bili v prvem koraku realizirani z uporabo hitrih TTL vezij srednje stopnje integracije, v letu 1987 pa že začnejo posamezni VLSI gradniki izrivati kompletna tiskana vezja. Prvi je bil seveda procesor. Razvoj je šel nezadržno naprej in tako se je konfiguracija kompletnega sistema v slabih desetih letih skrčila iz dvanajst (6 procesor in 6 pomnilnik) plošč tiskanega vezja (dimenzij cca 20 cm x 30 cm) na eno samo ploščo tiskanega vezja. Seveda je tudi zmogljivost sistema temu primerno narastla od 3.5 MIPS na 30 MIPS. Kakor se je izvedba spreminjala, je arhitektura v svojem bistvu ostala nespremenjena.

a) Zasnova procesorja.

Sestavlja jo osem enot povezanih s šestimi vodili. Slika 4.4.6 prikazuje to zasnovo.

b) Ukazna enota

I-enota (ukazna enota) krmili tok ukazov. Izvaja vejitvene ukaze, ravna z prekinitvami in pastmi. Generira tudi sistemsko uro. Izvajanje ukazov se prične, ko I-enota tvori naslov ukazov in ga dostavi ukaznemu predpomnilniku. Ta vrne ukaz in ga kot naslednji ukaz (NI) preko vodila NI pošlje vsem enotam. Tako je lahko dekodiranje ukazov in generiranje krmilnih signalov decentralizirano

c) Niz registrov

Niz registrov dostavlja vrednosti operandov za večino ukazov. Vsebuje 32 32 bitnih splošnih registrov. Niz registrov vsebuje tudi kopije 25 krmilnih registrov. Edini ukazi, ki imajo dostop do pomnilnika podatkov, so vlagalno/shranjevalni ukazi. Naslovi operandov teh ukazov se tvorijo na osnovi vsebine registrov in podatkov v samem ukazu. Niz registrov dostavlja vrednosti na dve vodili X (indeksno) in B (osnovno). Če je rezultat predhodnega ukaza potreben v naslednjem, se prenese mimo niza registrov (bolj točno: hkrati se vrši vpis v niz registrov in dostava vrednosti operanda v I ali E enoto).

d) Izvrševalna enota

Izvrševalna enota (E-enota) izvaja aritmetične izračune nad operandi in tvori naslove vlagalno/shranjevalnih ukazov. Vsebuje 32-bitno aritmetično-logično enoto, krožni pomikalnik in kompleksna vezja za maskiranje in zlivanje bitnih nizov. Na vsakem vhodu ALE je tudi predpomikalnik. ALE dostavlja rezultate preko R vodila nizu registrov. Če je rezultat naslov vlagalno/shranjevalnega ukaza, ga dostavi na CADR vodilo (uporabijo ga lahko krmilnik predpomnilnika, D-cache in TLB). Tvori tudi pogojno kodo na osnovi rezultatov operacije. Dostavi jo I-enoti, ki jo uporablja za pogojne vejitve in preskoke.

e) TLB, predpomnilnik, PP (angl. CACHE) in koprocesor.

TLB (Translation Look Aside Buffer) krmili dostop do virtualnega pomnilnika. Arhitektura podpira obseg naslovnega prostora, ki ga omogoča 64 bitov. Tako velik obseg naslovnega prostora omogoča enostavnejše upravljanje s pomnilnikom in tudi v primeru večopravilnih in večuporabniških sistemov ne zahteva posebne aparaturne in programske opreme za prehod iz enega procesa v drugega (ni potrebno prazniti vsebine predpomnilnika).

TLB izvede prevod virtualnega naslova v fizični naslov. Pri tem poskrbi tudi za zaščito in nadzor. Odvisno od izvedbe vsebuje različno število naslovov strani (od 4096 naprej), kar omogoča, da je le redko potrebno izračunati fizični naslov iz virtualnega.

Krmilnik predpomnilnikov upravlja dva predpomnilnika: za podatke (D-cache) in ukaze (I-cache). Upravljanje predpomnilnikov ni transparentno za PO, ampak so na voljo ukazi, ki omogočajo in zahtevajo eksplicitno rokovanje s predpomnilnikoma. To je omogočilo njuno učinkovito ločitev in podvojilo pasovno širino.

Koprocesor deluje paralelno z I-enoto in E-enoto in izvaja operacije v pomični vejici. To omogoča preostalemu delu procesorja, da procesira nekaj ukazov, preden je izračunan rezultat operacije v pomični vejici.

f) Vpliv arhitekture

Ker je arhitektura relativno enostavna in regularna, omogoča krmiljenje brez mikrokode. Vsi ukazi so dolgi 32 bitov, kar omogoča preprosto dekodiranje, ki je vrh vsega porazdeljeno na enote, da se zmanjša zakasnilni čas in kompleksnost (dolžina logične poti) dekodirnika.

Vsak ukaz se prične s preddostavo ukaza iz predpomnilnika in preddostavo operandov iz niza registrov. Od tu naprej so ukaz in operandi na voljo preko kratkih, paralelnih podatkovnih poti ostalim delom procesorja. V E-enoti delujeta ALE in

krožni pomikalnik z maskirno/zlivalnim vezjem popolnoma vzporedno in ne vplivata drug na drugega. Tudi I-enota lahko izračunava naslov ciljnega ukaza pri vejitvi vzporedno z E-enoto. To omogoča, da se vejitveni ukaz na osnovi pogoja-rezultata aritmetično-logične operacije izvrši v enem samem ciklu.

Drug primer izkoriščanja paralelnosti sta TLB in PP (Cache). Ker lahko virtualnemu naslovu pripada le en fizični naslov (če mu), poteka dostop do TLB in PP paralelno. Predpomnilnik poleg podatka ali ukazov dostavi tudi zaznamke o fizičnem naslovu. Ti se primerjajo s fizičnim naslovom, ki ga je dostavil TLB PP-ju preko PPN-vodila (physical page number). V večini sistemov ti operaciji potekata zaporedno. V HP arhitekturi pa sta možni tudi paralelno, ker sta oba predpomnilnika zelo velika in zagotavljata visok odstotek zadetkov.

g) Tekoči trak v procesorju

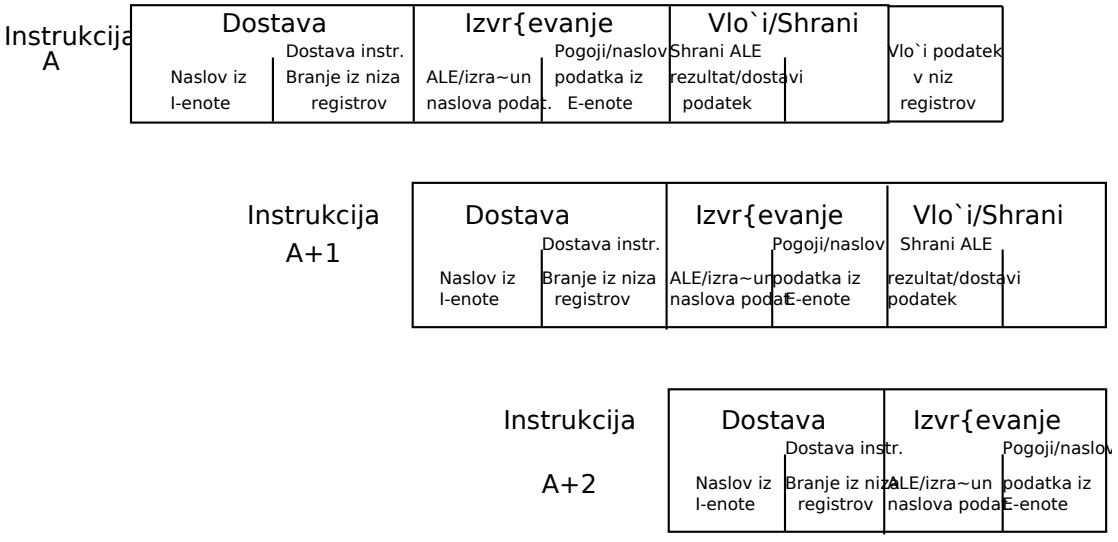
Tekoči trak ima tri stopnje: dostavno, izvršilno in vlagalno/shranjevalno.Vsaka stopnja je deljena v dve podstopnji. V prvi polovici dostavne stopnje poteka pošiljanje naslova ukaza v I-cache, v drugi polovici pa se ukaz že vrne in prične dekodiranje ukaza. Operandi, ki so v registrih se preberejo iz niza registrov.

V prvi polovici druge stopnje se izvrši aritmetična operacija ali pa se izračuna naslov, če je to vlagalno/shranjevalni ukaz. Če je to vejitveni ukaz, se izračuna tudi vejitveni naslov.

V drugi polovici se izračuna pogoj za vejitvene ukaze ali pri vlagalno/shranjevalnih ukazih dostavi naslov podatka v krmilnik predpomnilnika ali v TLB. Če je ukaz vejitveni, se izračunava vejitveni naslov med dostavo naslednjega ukaza. Če je pogoj izpolnjen pomeni to v večini arhitektur izgubljen cikel. V HP arhitekturi se za vejitveni ukaz skuša izvršiti ukaz, ki ni odvisen od pogoja.

Tretja stopnja je vlagalno/shranjevalna. V tej stopnji se podatek dostavi iz PP-ja ali vloži v PP. V resnici se v prvi polovici rezultat E-enote vpiše v register, v prvi polovici naslednje enote pa vloži (iz PP-ja) v niz registrov. Z nizom registrov se vpis izvrši v prvi polovici, ker je v drugi polovici nujno branje operandov ukazov, ki je v dostavni stopnji.

Aktivnosti v tekočem traku procesorja prikazuje slika 4.4.7.



Slika 4.4.7. Aktivnosti tekočega traku procesorja HP.

h) Zasnova PP-ja in TLB

Oba poskrbita, da so pogosto uporabljeni podatki in programi v zelo hitrem pomnilniku. TLB še dodatno poskrbi, da se prevajanje naslova izvrši prav tako v hitrem pomnilniku.

Predpomnilnik je razdeljen v 64K-bajtni ukazni in 64K-bajtni podatkovni (danes so že izvedbe s 128K-bajti). Oba sta razdeljena v 4096 16-bajtnih blokov. Vsak blok ima naslovni zaznamek, ki določa blok v pomnilniku iz katerega izvira. Če želi procesor podatek ali ukaz, se blok 16 bajtov kopira iz glavnega pomnilnika v PP. Vsi nadaljni nastopi se nanašajo na PP in blok se vrne v glavni pomnilnik le, če je bil spremenjen.

Podobno TLB pospešuje prevajanje, ker deluje kot PP za najbolj sveža prevajanja. Oba: virtualni pomnilnik in fizični pomnilnik sta razdeljena na strani dolžine 2K-bajta, in vsak vstop v TLB preslika virtualni naslov v fizični naslov strani. Vsak virtualni naslov sestavljata virtualni del (virtualni naslov strani) in fizični del (odmik znotraj strani). TLB prevede virtualni naslov strani v fizični naslov strani in ga združi s fizičnim naslovom znotraj strani, da nastane popolen naslov.
Da lahko realizira velike, hitre PP-je, HP arhitektura prepoveduje delitev naslovov, torej ne moreta dva virtualna naslova kazati na isto fizično lokacijo.

i) Operacija TLB

Virtualni pomnilniški prostor je razdeljen na strani dolžine 2048 bajtov. Podatkovna struktura v glavnem pomnilniku, ki jo imenujemo tabela strani vsebuje informacijo o vsaki virualni strani, ki je trenutno v uporabi (njena kopija obstoji v fizičnem pomnilniku-lahko tudi sekundarnem). Ta tabela ima polje za vsako virtualno stran in to polje vsebuje informacijo o zaščiti te strani in fizični naslov te strani (kje je v fizičnem pomnilniku).

TLB deluje kot PP za te informacije. Tako kot I-cache in D-cache vsebujeta kopije sveže uporabljenih pomnilniških lokacij, tako TLB vsebuje informacijo o sveže uporabljanih straneh. Obseg sega od 2048 strani za podatke in 2048 strani za ukaze do 4096 za vsako.

Za vsak dostop TLB preveri zaščito in poda fizični naslov te strani. Naslov pošlje PP-ju tako, da lahko ta preveri, če je iskana stran v predpomnilniku. V primeru kršene zaščite pride do pasti.

Pogreške v TLB-ju obravnava programska oprema. Če noben od vpisov v TLB ne ustreza virtualnemu naslovu, se generira past. Programska oprema iz tabele strani dostavi informacijo-vnos v TLB in ponovno izvrši ukaz.

TLB deluje na tekočem traku:

- V prvi polovici cikla prebere zaznamek in prevod iz TLB-ja, za ukaz, ki se dostavlja. V drugi polovici cikla preverja, ali se zaznamek ujema in prav tako izvede preverjanje zaščite (nadzor).

- V drugi polovici cikla se prav tako zaznamek in preveden naslov dostavita iz TLB-ja za dostop do pomnilnika za ukaz, ki je pravkar v izvršilni stopnji. Ta dostop se preverja v prvi polovici naslednjega cikla (ko je ukaz v vlagalno/shranjevalni stopnji). V vsaki polovici cikla štarta torej nov prevod naslova.

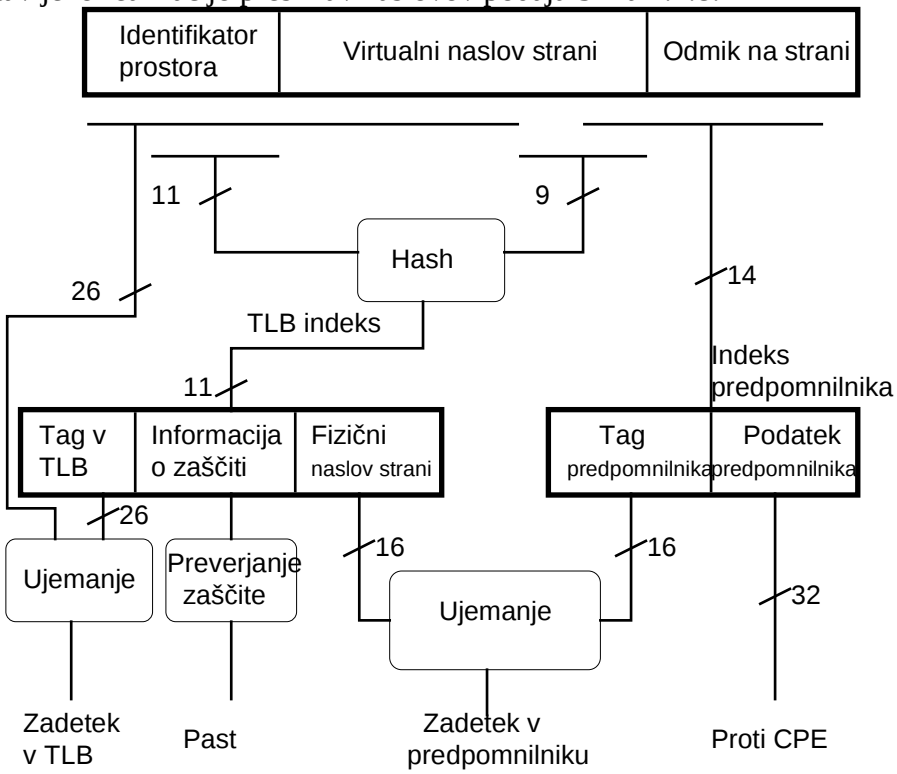
TLB je zasnovan na direktni preslikavi. To nam zagotavlja, da se lahko vsak naslov nahaja le na enem vnosu v TLB-ju. Če program želi drugo stran, ki se preslika

na isto mesto vnosa, izrine nova stran prejšnjo (premetavanje!). Čeprav je v takšni izvedbi odstotek pogreškov večji kot pri prosti preslikavi, lahko direktna preslikava bistveno skrajša cikel in poenostavi rokovanje s pomnilnikom.

Da ne bi prihajalo do nepotrebnega izrivanja sosednjih virtualnih strani, je uporabljen poseben princip razprševanja naslova. TLB naslavlja 9 LSB virtualnega naslova strani razpršenih z 11 LSB identifikacije prostora. Dva LSB naslova prostora se ohranita, preostalih devet LSB naslova prostora se zrcali in tvori bitno ekskluzivno ali operacijo z devet LSB virtualnega naslova strani. Dodatno pa še en naslovni bit ločuje med ukazom in podatkom, ker je TLB razdeljen na dva dela (pol vnosov za podatke, pol vnosov za ukazov), uporablja pa isto AO za preverjanje.

Poleg informacije o fizičnem naslovu vsebuje vsak vnos v TLB še informacijo o zaščiti (kaj je dopustno in kateri nivo privilegijev je potreben). Identiteta dostopa se mora ujemati z eno od štirih identitet v krmilnih registrih.

Ker TLB ne vsebuje prevodov za vse strani v pomnilniku, mora biti rešen problem prevoda za strani, ki niso v TLB. Če je stran v glavnem pomnilniku, se vstavi v TLB in ponovi izvajanje ukaza. Če strani ni v glavnem pomnilniku (je pa na disku), se shrani okolje procesa in štatrta program za dostavo v glavni pomnilnik. Poenostavljeno realizacijo preslikav naslovov podaja Slika 4.4.8.



Slika 4.4.8. Poenostavljena realizacija preslikav naslovov.

j) Operacija predpomnilnikov

Tudi pri predpomnilniku je uveljavljen tekoči trak. Ko se bere TLB, se bere tudi PP in ko se preverja zaščita, se preverja zaznamek PP-ja. Le pomnilnika za ukaze in podatke nista v tekočem traku kot pri TLB-ju, ampak ločena. Delujeta z dvakrat nižjo frekvenco in omogočata uporabo večjih pomnilnikov. Tudi za predpomnilnika velja princip direktne preslikave.

HP aritektura omogoča nekoliko enostavnejša predpomnilnika, ker PO skrbi za konzistenco med I-cache-jem in D-cache-jem in z V/I operacijami. Arhitektura

zagotavlja ukaze za čiščenje in prelivanje, ki omogočajo PO, da je kopija v pomnilniku ažurirana.

4.4.5 IBM RISC System/6000

Sestavlja ga 8-10 integriranih vezij po meri v VLSI tehnologiji. Odvisno od števila porabljenih vezij za podatkovni predpomnilnik (2 ali 4) dobimo dve osnovni različici: zmogljivejšo in manj zmogljivo.

Arhitektura striktno realizira tri koncepte:

- množica ukazov dela z registri,
- izbrano je ožičeno krmiljenje in
- tekoči trak.

K tem za RISC arhitekture običajnim izhodiščem so dodali še koncept superskalarnosti. V enem ciklu se dodeli v izvajanje več ukazov (maksimalno štiri in minimalno tri). Vrh pomnilniške hierarhije je deljen v dva dela: podatkovni in ukazni. Proceor ima posebno enoto za procesiranje vejitev. Skozi to enoto gredo vsi dostavljeni ukazi, vejitve izloča in jih procesira vnaprej. To omogoča, da tekoči trak enote za operacije v stalni vejici in enote za operacije v pomični vejici, delata v načinu nič ciklov za vejitve.

Nizkonivojski paralelizem zahteva bogate podatkovne poti. Tako je vodilo med pomnilnikom in podatkovnimi predpomnilniki široko štiri besede. Enako velja za vodilo med pomnilnikom in ukaznim predpomnilnikom. Vodilo med enoto za operacije v stalni vejici in med podatkovnim predpomnilnikom je široko eno besedo, tisto za enoto za operacije v pomični vejici pa dve besedi.

V množici ukazov poseben ukaz zmnoži in sešteje operande v pomični vejici:

a x b + c

zahteva za izvršitev en sam cikel. Množica vsebuje tudi močne ukaze za delo z nizi, poleg večkratnih pomikov še rotacijo vsebine. Dodana je še posebna oblika vlagalno/shranjevalnih operacij z ažuriranjem (podobne avtoinkrement ali avtodekrement).

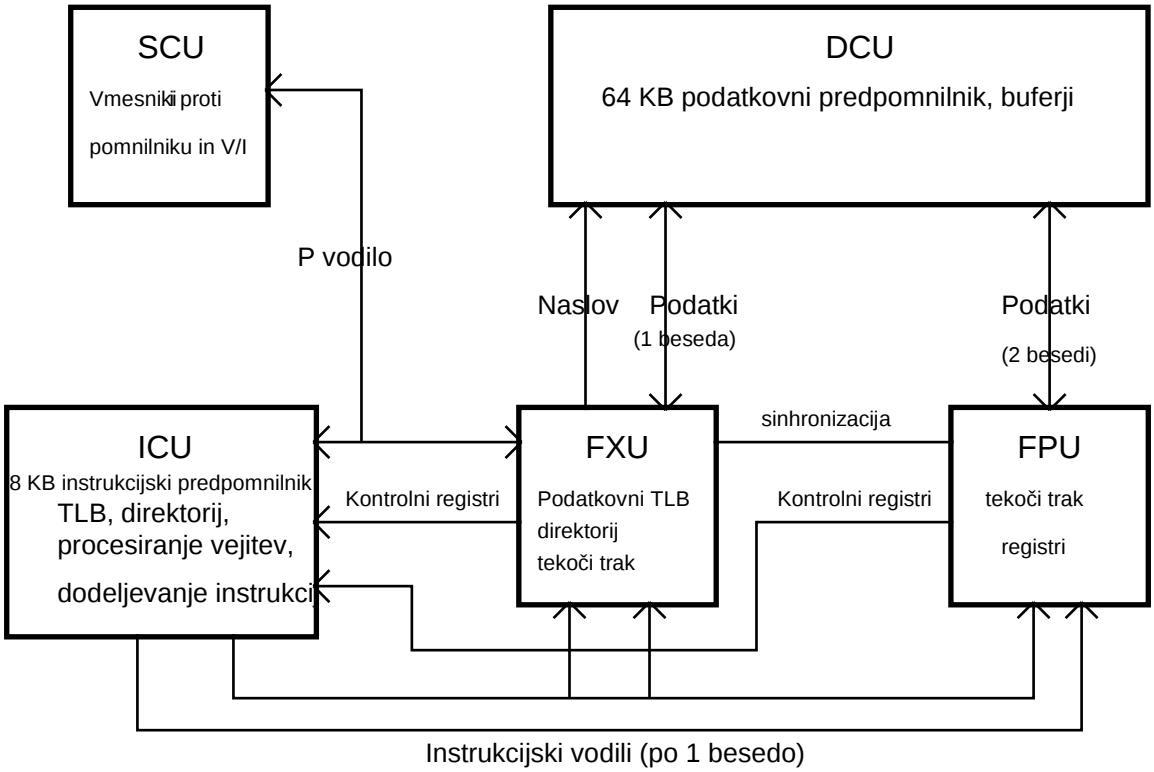
Arhitektura vključuje - kot smo že omenili - dva ločena predpomnilnika. Podatkovni predpomnilnik deluje na osnovi preslikave v množico pripadajočih naslovov (množica ima štiri elemente), enako velja tudi za ukazni predpomnilnik, pri tem ima množica dva elementa. Vse tri izvršilne enote vsebujejo še dodatne kratke predpomnilnike-vrste. Delovanje podatkovnih in i ukaznih predpomnilnikov je podprto z ločenima TLB. 32 bitni besedi je dodanih 7 bitov za ECC (detekcija dvojne in korekcija enojne napake) in dodaten rezervni bit. Mesto rezervnega bita ni fiksno, ampak se na novo določi ob vsakem vklopu. Občasno, vendar dovolj pogosto se avtomatično bere vsebina predpomnilnikov, izvrši ECC, loči mehke in trde napake in po potrebi izloči okvarjen bit. Tako je z veliko verjetnostjo vnaprej preprečeno, da bi prišlo do dvojnih napak.

Ker ima le izvršilna enota za vejitvene ukaze popolen pregled nad dostavljenimi in distribuiranimi ukazi, je možno zagotoviti učinkovito sinhronizacijo vseh treh izvršilnih enot. To omogoča, kjub izvrševanju več ukazov hkrati, natančne prekinitve. Poenostavljeno organizacijo RISC System/6000 (tudi America) prikazuje Slika 4.4.9.

a) Ukazni predpomnilnik in procesna enota vejitvenih ukazov:

- vsebuje 8 KB ukazni predpomnilnik z dolžino vrstice 64 bajtov. Vsebuje tudi imenik in TLB z 32 vnosi.
- Enota procesira vejitvene ukaze, ukaze za delo z registrom pogojev in ukaze-klice nadzornika. Dodeljuje ukaze ostalima dvema izvršilnima enotama.

Tako se lahko izvršujejo hkrati štiri ukazi: dva interno in po en v enoti za stalno in pomično vejico.



Slika 4.4.9. Poenostavljena organizacija RISC System/6000.

Prevajalnik skuša ukaze, ki postavljajo register pogojev, uvrstiti čimprej (čim so na voljo operandi) v niz ukazov. Arhitektura vsebuje register pogojev (CR), ki ima več polj. Vsaka izvršilna enota lahko vpiše svoj izračunan pogoj v različna polja. Procesor ve vnaprej za vejitev, kateri ukaz v vrsti ali tekočem traku drugih dveh enot bo postavil polje v CR, in lahko takoj po postavitvi izvrši dostavo iz pravilnih lokacij. Politika dostave je taka, da se vrši dostava ob predpostavki neizpolnjenega pogoja z nekaj izjemami. Med nje šteje n.pr. konstrukt-zanka z vejitvijo na koncu ali pa relativna vejitev z negativnim odmikom, kjer se vrši dostava ob predpostavki izpolnjenega pogoja. Poseben slučaj nastopi, kadar je vejitven naslov določen z vsebino registra, ki se bo šele izračunala. Ker imajo vsi registri takoimenovan veljaven bit, je procesna enota za vejitvene ukaze zopet promptno obveščena o izračunanem naslovu in lahko prične dostavo. Zaustavitve tekočega traku tako trajajo v najbolj neugodnem primeru le tri cikle.

b) Procesna enota za operacije v stalni vejici

Izvaja vse operacije v stalni vejici (množi in deli sta prava ukaza) ter vlaganja in hranjenje operandov v pomični vejici. Dostave v/iz podatkovnega predpomnilnika se lahko prekrivajo z ukazi tipa register-register. Vsebuje segmentne registre, podatkovni TLB s 128 vnosi (preslikava v množico z dvema elementoma) za preslikavo naslovov, zaščito strani in blokiranje podatkov. Blokiranje I-TLB, zamenjave v D-TLB in žuriranje tabel strani izvaja aparaturna oprema te enote. Tudi imenik podatkovnega predpomnilnika in krmiljenje tega pomnilnika sta vsebovana v tej enoti. Rezultati operacij lahko počakajo na vpis v podatkovni predpomnilnik v shranjevalnih baferjih.

c) Procesna enota za operacije v pomični vejici

To ni pravi koprocesor, saj je njeno delovanje tesno povezano z delovanjem ostalih dveh procesnih enot in tudi sinhronizirano. Daje rezultat v vsakem ciklu. Latenca tekočega traku je le dva cikla. Prevajalnik mora zato med dva podatkovno odvisna ukaza vedno vstaviti vsaj enega podatkovno neodvisnega. Kombinirani ukaz (a x b + c) ima prednosti, ker je zaokroževanje šele po seštevanju. Ta enota uporablja tudi princip preimenovanja (mapiranja) registrov. Procesna enota za operacije v stalni vejici lahko vloži vrednost v mapiran register, če je zahtevan register še vedno zaseden z ukazom, ki se izvaja v procesni enoti za pomično vejico. Aparaturna oprema podpira to mapiranje. Ima 64 registrov, 6 registrov, ki se lahko preimenujejo in dva posebna registra za deljenje. Omogoča tudi aparaturno paralelno ponormalizacijo rezultatov.

d) Podatkovni predpomnilnik

Vpis v podatkovni predpomnilnik ne povzroči tudi vpisa v pomnilnik (vpiši nazaj). Šele ko se onesnažena vrstica zamenja z novo, se vsebina vpiše v pomnilnik. Če je prenosna pot trenutno zasedena za vlaganje, se vsebina vpiše začasno v shranjevalni bafer. Obratno situacijo imamo pri pogrešku podatkovnega pomnilnika. Vsebina iz pomnilnika se ne naloži direktno v predpomnilnik (to bi zahtevalo štiri cikle), ampak v vlagalni bafer in od tam naknadno v predpomnilnik. Uporaba obeh baferjev bistveno dvigne procent zadetkov in zmanjša čas čakanja procerjev.

Velika dolžina vrstice podatkovnega predpomnilnika zagotavlja neke vrste dostavo podatkov vnaprej. Dolga vrstica s prekrivanjem dostopa in prenosa predhodnega podatka 100% izkorišča pasovno širino pomnilnika.

4.5. Mikroprocesor HP Alpha 21064.

4.5.1. Uvod

Proti koncu 80. let se je Digital lotil razvoja procesorja za 21. stoletje. Nastal je 64-bitni mikroprocesor v RISC arhitekturi, ki deluje z 200 MHz. Na silicijevi ploščici 1,4 * 1,7 cm so v CMOS tehnologiji realizirali v 0.75 mikronski tehnologiji in s triplastno metalizacijo 1,68 miljoni tranzistorjev. Prav gotovo je procesor DEC 21064 ali DEC Alpha AXP eden vodilnih na področju 64-bitnih mikroprocesorjev (kot so Power PC, SPARC, MIPS ipd). Mikroprocesor je nekaj več kot dve leti načrtovalo več kot 30 inženirskih skupin v 10 državah.

Cilj načrtovalcev je bil zasnovati mikroprocesor tako, da bi v prihodnjih 25 letih pridobil 1000-krat na zmogljivosti. V zadnjih 25 letih se je namreč sposobnost računalnikov povečala za 1000-krat. Najprej je sledila odločitev, da bo arhitektura 64-bitna s skrčenim naborom ukazov (RISC). Urni takt naj bi se v 25 letih povečal za

faktor 10. Tako so prva vezja Alpha delovala z uro 125 MHz, zdaj pa že z 275 MHz. Ko ure ne bo več mogoče pohitriti, bo treba narediti več opravil v časovni enoti. Zato je bila Alpha načrtovana tako, da lahko hkrati v enem urnem taktu izvaja do 10 ukazov. Posamezna izvajanja ukazov pa se med seboj lahko prekrivajo (pipeline). To je gotovo ena od lastnosti, ki mikroprocesor ločijo od drugih arhitektur RISC. Naslednjo desetkratno povečanje hitrosti pa so načrtovalci predvideli z vzporednim dodajanjem več procesorjev. Tako naj bi en sistem vseboval tja do 10 mikroprocesorjev, ki bi si delili skupni pomnilnik. V ta namen so že sedaj dodali take ukaze, ki podpirajo večprocesorski sistem. Vse posebnosti operacijskih sistemov VMS in Unix so realizirali s posebnimi klici funkcij iz knjižnice sistemskih privilegiranih (pod)programov (PAL code). Razvili so tudi poseben postopek, da so strojno (dvojiško) kodo ukazov VAX preuredili v kodo za Alpho in pri tem ni bilo treba prilagajati oziroma spreminjati arhitekture.

4.5.2. Arhitekturne značilnosti

4.5.2.1. Alpha AXP je običajna **RISC arhitektura** za vpis in izpis v pomnilnik oziroma registre. Vse operacije se delajo le med vrednostmi v registrih. Poleg operacije s celimi števili je podprto še računanje s plavajočo vejico v formatu VAX oziroma IEEE. Predvideva se, da se bo čim več ukazov izvajalo s prekrivanjem, da bo čakanja na izvršitev (predhodnega) ukaza kar najmanj in da bodo vsi podatki v pomnilniku poravnani (to pomeni, da imajo vsi podatki dolžine 2**N bajtov spodnjih N bitov po vrednosti "0").

4.5.2.2. Alpha AXP uporablja **64-bitni linearni naslovni** prostor. Registri, naslovi, cela števila ter števila v plavajoči vejici in nizi se vsi obravnavajo kot 64-bitne veličine. Mikroprocesor tudi **nima** segmentacije pomnilnika!

4.5.2.3. **Registrski niz** je bil načrtovan tako za hranjenje celih kot števil v plavajoči vejici. Odločili so se za dvojna niza po 32 64-bitnih registrov. To pa predvsem zato, ker se lahko v 32 registrih shrani najmanj 8 neodvisnih izračunov. Več registrov pomeni tudi večji prostor na ploščici silicija, zato pa upočasnitev celotnega vezja.

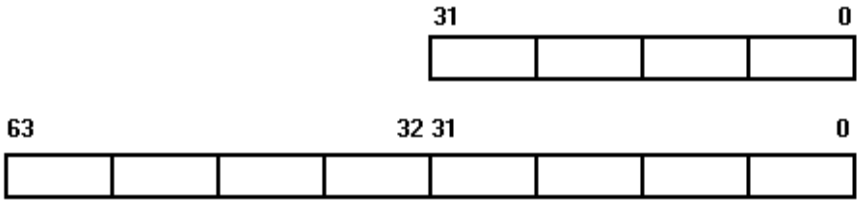
4.5.2.4. **Hkratno procesiranje** (multiple issue) ukazov je prav gotovo vplivalo na načrtvanje mikroprocesorja, kajti arhitektura Alpha nima registra stanj, omogočanja prekinitev, večjih registrov za množenje oziroma deljenje, ukazov za preskok (skip), pasti in prekinitev, ki so vezane na izvajanje aritmetičnih ukazov, in ukazov za zapis posameznega bajta v pomnilnik. Vsi ti ukazi, ki jih včasih najdemo v drugih arhitekturah RISC, ne omogočajo hkratnega izvajanja več ukazov. Namesto naključnih pasti, do katerih lahko pride pri izvajanju aritmetičnih operacij, so privzeli rešitev iz računalnika CRAY-1, ko se z ukazom TRAPB (trap barrier) nadzorovano - programsko - ugotovi, ali je prisotna prekinitev (past) ali ne.

4.5.2.5. **Porazdeljena uporaba pomnilnika** je vgrajena v vsakem ukazu, ki dela s pomnilnikom. Vrstni red akcij je podoben kot pri procesorju MIPS R4000. Zaporedje večkratnih zapisov in izpisov z enega procesorja pa se rešuje tako, da se vstavi ukaz MB (memory barrier), podobno kot je to narejeno pri računalnikih IBM System/360.

Tabela 4.5.1 Primerjava lastnosti nekaterih mikroprocesorjev

	K5	SPARC	Power PC 620	T5	DEC 21064
arhitektura	32-bit CISC	64-bit RISC	64-bit RISC	64-bit RISC	64-bit RISC
hkratnost izv. uk.	4	4	4	4 (5)	od 2 do 10
prekrivanje uk.	5 stopenj	9 stopenj	6	5	do 4
primarni predpomnilnik	16 kB / ukazi 32 kB / pod.	16 kB / ukazi 16 kB / pod.	32 kB / ukazi 32 kB / pod.	32 kB / ukazi 32 kB / pod.	8 kB / ukazi 8 kB / pod.
asociativnost	4-kratna skup.	2-kratna skup.	-	2-kratna skup.	polna
ura [MHz]	100-150 (1996)	do 200	133	200	125-275 (1994)
št. tranz., velikost	4.1 mio; 0.5μm	-	7 mio	6 mio; 0.35 μm	od 1.7 do10 mio; 0.5μm
del. napetost, tehnologija	3.3V; CMOS	3.3V; CMOS	3.3 V; CMOS	3.3 V; CMOS	3.3 V do 1,8V; CMOS
št. vzpor. enot	5	9	6	5	5
naslavljanje	linearno, s pom. prepletanjem	pom. prepletanje	pom. prepletanje (40 bit)	pom. prepletanje (44 bit)	linearni naslov (od 34 do 64 bit)
št. reg.	8	8*24	-	2*64	2*32
proizvajalec	AMD	Texas, Sun	Apple, IBM, Mot.	Mips	Digital

4.5.2.6. Za **predstavitev podatkov** se uporablja 32-bitni, oziroma polni 64-bitni zapis. Vsi ukazi so v 32-bitni kodi in običajno delajo nad 64-bitno vsebino. Le redki ukazi dajo kot rezultat 32 bitov. Vseh 64 bitov lahko hkrati zapisujemo v pomnilnik, glavni razlog za 32-bitne operacije pa je združljivost z drugimi (VAX) računalniki.



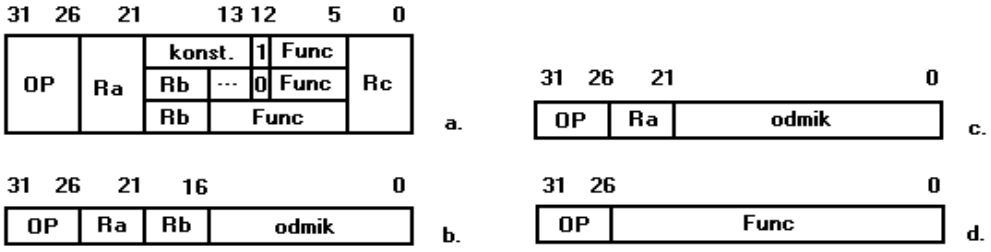
Slika 4.5.1 Osnovna podatkovna tipa mikroprocesorja Alpha AXP, ki se uporabljata tako za zapis celih števil kot tistih zapisanih v plavajoči vejici v formatu VAX ali IEEE

Bajt in beseda (16 bitov) nista osnovna tipa. Obdelujemo ju lahko z običajnimi ukazi za 64-bitno aritmetiko in ukazi za delo z bajti.

4.5.2.7. **Naslavljanje** v mikroprocesorju Alpha je **bajtno,** vpis in izpis iz pomnilnika pa sta ali 32- ali 64-bitna. Bajti so poravnani od zgornjega navzdol (big-endian, najvišji bajt na najnižjem mestu) ali od spodnjega navzgor (little-endian, najnižji bajt na najnižjem mestu). Naslov mora biti vselej poravnan, sicer se nastavi past (alignment trap). Navidezni naslov je 64-biten, pri čemer mora biti različnih najmanj 43 bitov. Virtualni naslovi se s pomočjo operacijskega sistema preslikajo v fizične pomnilniške naslove. Uporabljajo se strani, ki imajo vse isto dolžino, na primer 8, 16, 32 ali 64 KB. Poseganje v zaščito strani povzroči past. V večprocesorskih sistemih je pravi (fizični) naslov skupen vsem procesorjem.

4.5.3. Ukazi Alpha 21064

Mikroprocesor Alpha AXP ima 4 vrste ukazov: aritmetično/logične, pomnilniške, skočne in klice privilegirane programske knjižnice (CALL_PAL).



Slika 4.5.2 Formati ukazov: a) aritmetično/logični, b) pomnilniški, c) skočni ukazi in d) klic privilegirane knjižnice

Vsi ukazi so 32 bitni: 6 bitov pomeni kodo ukaza, 5-bitna so polja, ki označujejo registre Ra, Rb in Rc. Drugi biti označujejo funkcijo, vrednost ali odmik. Zaradi preproste uporabe registrskega niza se v Rb nikoli ne zapisuje, iz Rc pa ne bere.

4.5.3.1. Vsi **aritmetični in logični ukazi** so oblike:

$$Rc = Ra\ op\ Rb.$$

Operacija *op* se izvaja vedno le med registroma Ra in Rb, rezultat pa gre v tretjega (Rc). Pri celoštevilčnih ukazih koda in 7 bitov funkcijskega polja natančno določajo operacijo, pri operacijah s plavajočo vejico označuje funkcijo 11 bitov.

Ukaze lahko združimo v pet skupin: ukazi za celoštevilčno aritmetiko, logični ukazi, ukazi za ravnanje z bajti in ukazi za računanje s števili v plavajoči vejici in mešani ukazi.

Tabela 4.5.1 Kode ukazov

fADD	AND	CMPBGE	ADDf	CALL_PALL
fSUB	BIC (OR)	EXTb	SUBf	FETCH
CMPEQ	XOR	IMMSKbPb	MULf	FETCH_M

CMPLT	EQV	ZAP	DIVf	MB
CPMLE	ORNOT	ZAPNOT	CMPf	RPCC
CMPULT	SLL		FBf	TRAPB
CMPULE	SRL		CVTf	
MUL	SRA			
UMULH	CMOVf			

Opomba: f in b del kod sta vezana na delovni format!

Ukazi za celoštevilčno seštevanje, odštevanje in množenje lahko povzročijo past. Vsi ukazi delajo ali z 64-bitnimi ali pa z 32-bitnimi operandi. Ukaza za celoštevilčno deljenje ni! Deljenje s konstanto se izvede kot množenje z recipročno vrednostjo, deljenje s spremenljivko pa s pomočjo podprograma. Rezultat ukaza za primerjenje je vrednost "0" ali "1", ki se zapiše v ponorni register.

Ukazi za delo z bajti so namenjeni vpisu in izpisu manjšega niza (1, 2, 4 ali 8) bajtov. Velike nize pa prenašamo kot 64-bitne besede. Tudi ukazi za primerjanje delajo nad celotno dolžino, kar nam omogoča hkratno obdelavo 8 bajtov tako, da (na primer v jeziku C) dosežemo veliko hitrost procesiranja nizov.

Ukazi za računanje s plavajočo vejico so seštevanje, odštevanje, množenje, deljenje ter primerjanje in pretvarjanje iz enega formata v drugega. Prvi štirje ukazi uporabljajo tako 32-bitni kot 64-bitni format za VAX in IEEE standard.

4.5.3.2. Ukaza za **vpis in izpis** v pomnilnik sta dvooprandna. En del kode določa register Ra in (16-bitni) odmik. Drugi del pa naslov Rb, kamor se prišteje še odmik (z razširjenim predznakom). Tako dobljeni navidezni naslov se preslika v pravi (fizični) naslov.

Tabela 4.5.2. Ukazi za vpis in izpis:

LDA	LDL	STL
LDAH	LDL_L	STL_C
	LDQ	STQ
	LDQ_L	STQ_C
	LDQ_U	STQ_U

Ukazi za vpis in izpis so namenjeni zgolj premikanju podatkov iz pomnilnika ali vanj. Ker se med podatki ne izvajajo nobene operacije, tudi ne prihaja do pasti. Uporabljen podatek se naslavlja vedno poravnano.

4.5.3.3. **Skočni ukazi** vedno testirajo le register Ra. Odmik se z razširjenim predznakom prišteje k 64-bitnemu programskemu števcu PC, če je le pogoj izpolnjen.

Tabela 4.5.3 Ukazi skokov:

BEQ	BLBC	BR	JMP
BGE	BLBS	BSR	JSR
BGT	BLE		RET
	BLT		JSR_COROUTINE
	BNE		

Ukazi pogojnih skokov opazujejo celoštevilčno vrednost registra Ra, ki je lahko:

>0, >=0, =0, !=0, <=0, <0, soda ali liha.

Ukazi za plavajočo vejico so isti, le da tu ne opazujemo vrednost ali je liha ali soda. Brezpogojni skoki so izvedeni vselej tako, da se v Ra shrani naslov vračanja (klic podprograma). Če tega ne potrebujemo, postavimo Ra=R31!

4.5.3.4. **Klici programske knjižnice** (CALL-PAL) imajo 6-bitno kodo za ukaz in 26 bitov, ki povedo, kateri privilegirani program se bo izvedel. Z njimi rešujemo prekinitve, pasti, napake v delovanju stroja in arhitekturne posebnosti, podobno kot je to narejeno na osebnih računalnikih s klici funkcij BIOS. Zasnova mikroprocesorja Alpha AXP pa je taka, da kode PAL niso predvidene samo za pospešitev obstoječega operacijskega sistema (VMS oziroma OSF/1) ter prilagoditev arhitektur procesorjev VAX in Alpha, pač pa tudi za bodoče implementacije, na primer za večprocesorski sistem v realnem času, za povečanje zanesljivosti delovanja (preprečevanje vdorov v sistem) ipd.

4.5.4. Sklep

Arhitektura mikroprocesorja Alpha AXP je zasnovana tako, da jo je mogoče širiti, če se izkaže, da so zmogljivosti omejene. Trenutno je za navidezno naslavljanje predvidenih 43 bitov, za fizični naslov pa se uporablja le 34 bitov. Prav tako začetna enačica vezja dela s stranmi, ki so 8KB-tne, predvideva pa se, da bodo v končni enačici 64 KB-tne.

Nekaj kod, ki so še proste, je predvidenih za bodoče ukaze za izpis in vpis 128-bitnih vrednosti, za celoštevilčne operacije med njimi in take s plavajočo vejico.

Cilji, ki so si jih zastavili načrtovalci vezja DEC 21064 so doseženi, saj je vezje leta 1992 celo prišlo v knjigo rekordov kot najhitrejši mikroprocesor v enem integriranem vezju. Alpha teče z operacijskimi sistemi VMS, Unix in Windows NT. Prav tako pa se dajo preprosto prevesti in uporabiti vsi programi (v strojni kodi) z VAX-a in MIPS-a.

4.6. Mikroprocesor IBM PowerPC.

4.6.1. Uvod mikroprocesorji družine IBM PowerPC 740/750

Kratica »Powe« pomeni Performance Optimized with Enhanced RISC architecture, Optimizirana zmogljivost z izboljšano RISC arhitekturo. Nikakor se ne moremo znebiti občutka, da sta si arhitekturi Pentiuma in Power procesorjev zelo podobni. Morda jih celo delajo isti načrtovalci?!

Družina 750 uporablja 32-bitno zasnovo arhitektur Power, ki uporablja 32-bitno naslavljanje, celoštevilčno aritmetiko za 8, 16 in 32-bitne vrednosti in plavajočo vejico v 32 in 64-bitnem formatu. Mikrorprocesor 750 je superskalarni stroj, ki

- dostavi štiri ukaze hkrati,
- razpošlje 2 ukaza v obdelavo procesnim enotam v enem urinem taktu,
- obdeluje 6 ukazov hkrati v urinem taktu in
- izvede skoraj vse ukaze v enem taktu.

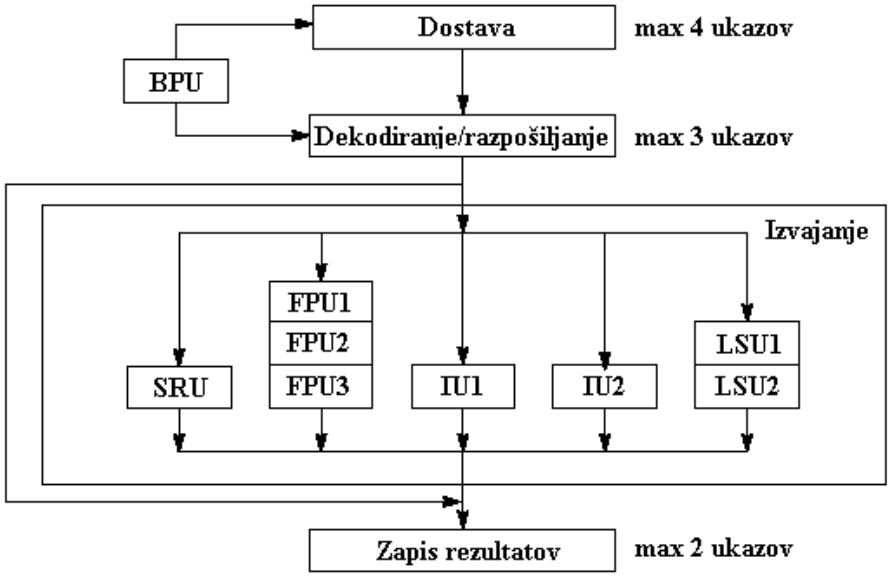
Uporablja 6 izvajalskih enot za

- plavajočo vejico (FPU), ki jo sestavljajo tri enote. Uporablja niz 32 64-bitnih registrov (FPR). Ima eno stopnjo za rezervacijo. Enota ima polje za množenje in seštevanje.
- procesirnje skokov (BPU). Brezpogojni in pogojni skoki, kjer je pogoj poznan, se rešijo takoj. Za nerešene skoke se določi skok s predikcijo. Enoto za predikcijo sestavlja 64 vhodov, organiziranih kot 4 vhodni, 16 skupinski PP (BTIC), ki hranijo (ponudijo) uporabljene skoke med izvajanjem programa. Enoto za sled in zgodovino uporabe skokov sestavlja 512-vhodna tabela (BHT), s po dvema bitoma za vstop v štiri nivoje predikcije.
- sistemske registre (SRU). Enota izvja razne sistemske operacije, pogojne skoke, prenose med registri. Ima enostopenjsko enoto za rezervacijo.
- vpis in izpis (LSU). Ima dve stopnji za rezervacije, tekoči trak do PP, svoj seštevalnik za izračun EA (efektivega naslova), poravnavo (alignment), tri stopnje za shranjevanje podatka, podpira debeli in tanki konec naslavljanja.
- računanje s celimi števili ima dve enoti, IU1, ki izvaja vse operacije in IU2, ki izvaja pravtako vse, razen množenja in deljenja. Uporablja niz 32 32-bitnih registrov (GPR). Imate enoto za rezervacijo, ki sprejme ukaz iz vezja za razpošiljanje (dispatch) in registrskega niza. Enoto sestavljo tri podenote za: seštevanje/primerjanje, logične operacije in pomikanje, ki vse svojo operacijo opravijo v enem urinem taktu. Opravlja prenose med GPR in FPR ter PP in pomnilnikom. Enota lahko izvede največ en dostop (čitanje) do pomnilnika preko vrste (out-of-order). Pisanja ne gredo preko vrste. Le-teh se izvede enega v urini enoti, tri urine cikle pa potrebujemo za zapis v podatkovni PP.

Sposobnost izvajati več ukazov vzporedno in uporaba preprostih ukazov daje stroju dodatno zmogljivost. Večina celoštevilčnih ukazov se izvede v eni urini periodi. Enota FPU ima tekoči trak: ukaz se razbije v tri podukaze, ki se izvajajo v treh stopnjah tekočega traku (cevovoda). Tipično se en ukaz v plavajoči vejici izvaja v eni stopnji, druge stopnje pa izvajajo druga dva. Torej se izvajajo tri ukazi v plavajoči vejici hkrati. Ukazi seštevanja v dvojni natančnosti rabijo dodatno še tri urine cikle, za množenje in množenje s seštevanjem pa štiri.

Na sliki 4.6.1. vidimo organizacijo procesnih enot v mikroprocesorju Power 750.

Mikroprocesor ima tudi vgrajen 32kB, osmero-smerni, 128-skupinsko asociativen predpomnilnik (harvardska arhitektura) za ukaze in podatke (vrstica v PP je 32-bajtna) in samostojni enoti za podatkovno in ukazno ravnaje (MMU). Vsaka od enot MMU ima svoj predpomnilnik (DTLB, ITLB), ki je organiziran kot 128 vhodni dvosmerni skupinsko asociativen PP in hrani zadnje uporabljene preslikave naslovov strani. Preslikave blokov gredo preko 4-vhodnega polja (IBAT, DBAT). Med preslikavo bloka se efektivni naslov primerja hkrati z vsemi štirimi vrednostmi v BAT.



Slika 4.6.1. Procesna superskalarna enota s tekočimi trakovi.

Predpomnilnik nivoja 2, L2, je pravtako vgrajen in je organiziran kot dvo-smerni skupinsko asociativen PP, z zunanjim vezjem DRAM. Le-ta je dostopen preko vrat (portov), ki podpirajo banko podatkov do 1MB. Vrstica je 64-bajtna pri 256kB ali 512KB PP in 128-bajtna pri 1MB PP.

Mikroprocesor 750 ima 32-bitna naslovna, 32-bitna ukazna in 64-bitna podatkovna vodila. Različne naprave na vodilih tekmujejo za vodilo preko posebnega vezja. Sistem ima tudi 3 stopenjski protokol skladnosti PP.

4.6.2. Formati ukazov in nabor

Ukaze lahko razvrstimo v 10 skupin:

a. celoštevilčni aritmetični ukazi (addx, addcx, divdx, mulhdx, subfx,…) uporabljajo dva formata.

	D	A	B	OE		Rc
	D	A	SIMM			

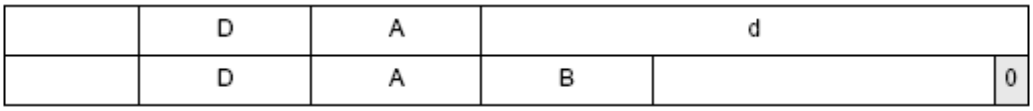
Slika 4.6.2. Format uporabljajo tudi drugi ukazi. Prazna polja so operacijske kode. A; B in D so imena registrov, SIMM je takojšnja vrednost.

b. primerjalni ukazi (cmp, cmpl, …) uporabljajo pravtako podoben format kot na Sl. 4.6.2.
c. Ukazi za logične funkcije (andx, extsbr, nandx, norx, orx, xorx, …), pomike (sidx, siwx, srawx, …) in rotacije (rldclx, rldcrx, …), uporabljajo format, kjer se označuje število pomikov, oziroma rotacij.

	S	A	sh	mb		sh	Rc
	S	A	SH	MB	ME		Rc

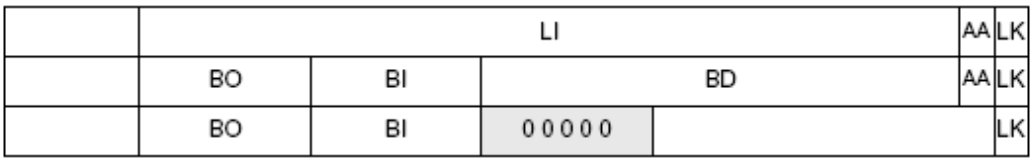
Slika 4.6.3. Format pomikov in rotacij.

d. Formati ukazov za plavajočo vejico (faddx, fmulx, fsubx, fsqrtx, fmaddx, fmaddsx, fcfdx, fcfidx, fcmpo, fcmpu, mcrfs, mffsx, …) so podobni onim za celoštevilčno računanje.
e. Format ukazov za branje (lbz, lbzu, lbzux, lhbrx, lmv, lswl, isync/sinhronizacija/, lfd, lfdu/za PV/, …) in vpis (stb, stbu, stbux, sthbrx, stmv, stswl, stfd, stfdu, fabsx /za PV/, …) iz pomnilnika je na Sl.4.6.4.



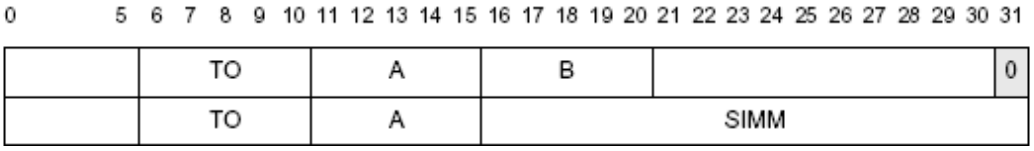
Slika 4.6.4. Format ukazov za pisanje in branje iz pomnilnika.
Polje d označuje naslov.

f. Skočni ukazi (bx, bcx, crand, creqv, cror, …) uporabljajo format na Sl. 4.6.5.



Slika 4.6.5. Formati skočnih ukazov.

g. Skupina ukazov za kontrolo procesorja, pasti, PP, naslovnega upravljanja ipd ima vec različnih formatov (36 ukazov). Podajamo primer formata za past.



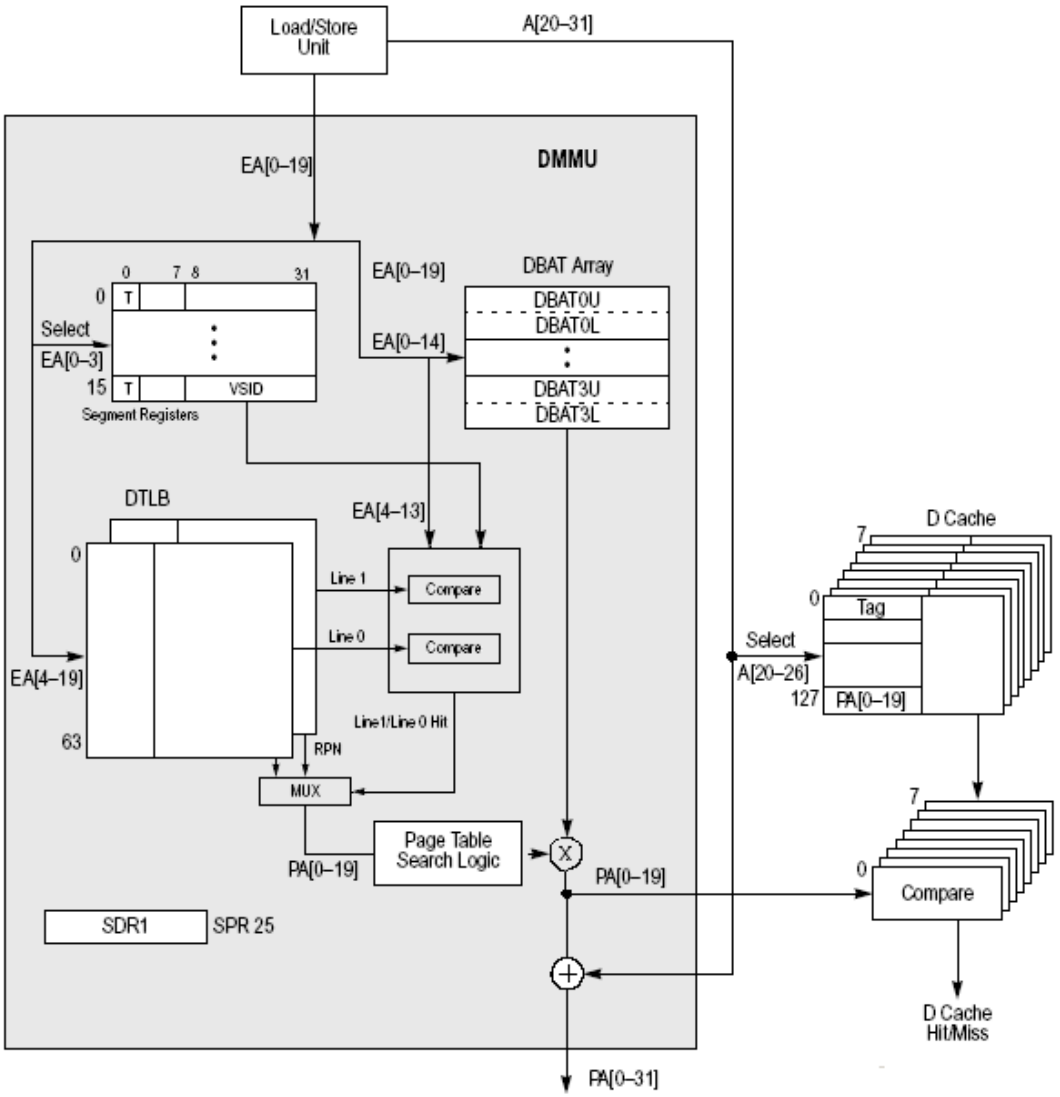
Slika 4.6.6. Format ukazov pasti. Kot vidimo IBM uporablja označevanje bitov od leve proti desni: bit 0 je na položaju MSB!

4.6.3. Enota za upravljanje (ravnaje) s pomnilnikom

Upravljanje s pomnilnikom (MMU enota /Memory Menagement Unit) v mikroprocesorjih Power PC je 32-bitna. Torej naslavlja do 4GB glavnega pomnilnika, strani so velike 4kB, segmenti pa 256kB. Poleg tega mirkroprocesor uporblja 52-bitni virtualni naslov in preslikovalne tabele v družini 32-bitnih naslovov ter poseben mehanizem (BAT, block translate mechanism) za preslikavo blokov pomnilnika, ki so veliki med 128kB in 256MB.

Osnovni hardver 750 MMU podpira:

- realni način naslavljanja, preslikava efektivnega naslova v fizični je možno izvajati ločeno za podatke in ukaze;



Slika 6.6.7. Podatkovna enota za upravljanje s pomnilnikom (DMMU). DBAT je enota za preslikavo blokov. SDR1 je register, ki določa dostopnost do strani.

- preslikavo blokov (BAT). Vsak od štirih vhodov v polje preslikav IBAT in DBAT omogoča preslikavo blokov velikosti do 256 MB iz prostora 32-bitnega efektivnega naslova v fizični pomnilniški prostor. Ta mehanizem se uporablja pri preslikavi zelo velikih blokov, kadar se njihov naslov ne menja prav pogosto;

- preslikavo segmentov (segmentiranje). Segmentiranje se lahko nadaljuje iz preslikvave blokov. Sicer pa ger na sledeč način: iz EA v VA ter iz VA v PA. 32-bitni efektivni naslov (EA) se razširi do 52-bitega virtualnega naslova, tako da se uporabi zgornjih 20 bitov EA[0:19] in preslika preko segmentnih registrov, 24-bitna vrednost VSID. 52-bitni virtualni naslov razdeljuje virtualni pomnilnik v (okvirje) strani velikosti 4kB, ki se preslikajo v fizični pomnilnik. Najtežji 4 biti (EA[0:3]) služijo kot indeks v niz 16-tih segmentnih registrov. Drugih 24-bitov se dobi iz ITLB ali DTLB tako, da se z biti EA[4:19] naslavlja ustrezni 2-smerni 64-vhodni TLB in primerja bite EA[4:13] ter tiste iz ene od dveh vrstic TLB z vrednostjo iz segmentnega registra. Zadetek nam da preslikanih 20 bitov fizičnega naslova okvirja strani, torej PA[0:19]. Biti PA[20:31] so biti naslova bajta znotraj strani.

Mikrorpocesor 750 vsebuje tudi še vezja, ki jih arhitektura PowerPC ne zahteva. To so:

- dva ločena preslikovalna predpomnilnika TLB, in sicer 128-vhodni (2*64), 2-kratni skupinsko asociativen ITLB in DTLB, ki hrani zadnje uporabljene strani;
- vezje za iskanje strani. Ko dobimo 52-bitni naslov se skuša dostaviti ukaz (PTE), ki vsebuje fizični naslov, iz TLB na čipu. Če se preslikave ne najde v TLB (pogrešek), hardver izvede »operacijo iskanja« strani;
- razvrednotenje in sinhronizacija TLB;
- množico ukazov za delo s SR, TLB, BAT in SDR1.

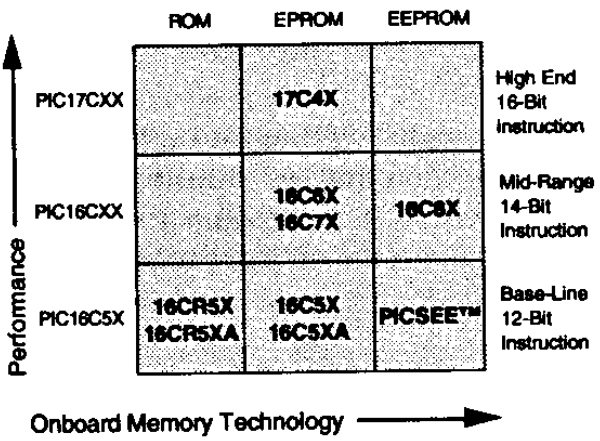
4.7. Mikoračunalniki, oziroma mikrokontrolerji

4.7.1. Uvod

V množici mikrokontrolerjev, oziroma mikroračunalnikov (v enem integriranem vezju), ki jih izdelujejo razne firme na primer AMD, Motorola, Texas in drugi smo izbrali proizvod firme Microchip zato, ker nudi ta trenutek največ. Mikrokontrolerji z oznako PIC 16/17 so kombinacija mikroprocesorjev sodobne arhitekture, velike zmogljivosti, nizke cene in so vsi v enem integriranem vezju. Veliko teh vezij se uporablja v avtomatizaciji in cenenih uporabniških napravah, računalniških vmesnikih, samostojnih napravah, telekomunikacijah ipd.

Mikrokontrolerji (Sl. 4.7.1.) PIC16CXX in PIC17CXX so v tehnologiji CMOS. Imajo 8 bitno centralno procesno enoto v arhitekturi RISC, 12-bitno ali 16-bitno ukazno kodo, poenostavljen nabor ukazov (RISC) in vgrajene vektorske prekinitve. Na primer vezje iz družine PIC17C42 je poznano v raznih aplikacijah krmilnih naprav. Vezje lahko deluje samostojno, lahko pa ukaze sprejema iz zunanjega pomnilnika, velikosti 64k 16-bitnih besed. Vezje ima vgrajen časovnik (timer) in možnost vhodno/izhodnega naslavljanja. Današnja vezja družine PIC 16/17 vključujejo še razširjen časovnik, analogno/digitalni (A/D) pretvornik razširjeni nabor ukazov za delo s pomnilnikom, medprocesorske povezave ter notranji ROM ali UV EPROM ali EE PROM.

Delo z mikrokontrolerji PIC je enostavno, saj dobi uporabnik programsko podporo, programator in tudi emulator (simulator vezja na gostujočem računalniku).



Slika 4.7.1. Družine mikrokontrolerjev.

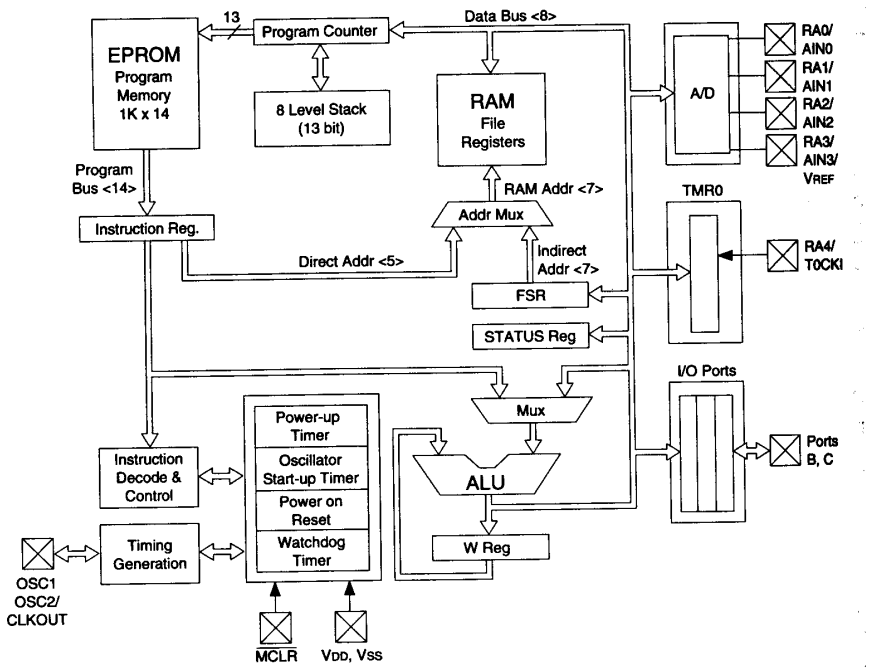
Mikrokontrolerji so opremljeni s posebnim serijskim CMOS EEPROM vezjem, velikosti 64k do 256k bitov. Lahko pa vsebujejo paralelna CMOS EPROM vezja od 64k do 512k bajtov.

4.7.2. Mikrokontroler PIC 16C71

Mikrokontroler PIC 16C71 je v tehnologji CMOS zmogljiv 8 bitni procesor. Vsebuje:

- vgrajeno vezje EPROM,
- 8 bitni analogno/digitalni pretvornik in 4 vhodne linije. Čas konverzije je 30μs,
- 14-bitni enobesedni ukaz, ki se izvede v 250 ns, oziroma z uro 16MHz,
- 4 prekinitvene linije,
- 8 nivojski hardverski sklad,
- 8-bitni časovnik, z 8-bitnim delilnikom (torej je to skupaj 16-bitni časovnik),
- 13 dvosmernih v/i linij z močnim izhodnim tokom (25 mA).

Kontroler programiramo z zbirnikom preko IBM PC ali združljivim računalnikom.



Slike 4.7.2. Blok shema PIC 16C71.

4.7.3. Arhitekturne značilnosti

Veliko zmogljivost PIC16C71 lahko pripišemo predvsem njegovi arhitekturi RISC. Mikrokontroler s Sl. 4.7.2. uporablja harvardsko arhitekturo, kar pomeni, da ima ukazna in podatkovna vodila ločena. Na ta način smo se izognili von Neumannovemu ozkemu grlu, ko se podatek in ukaz dostavljata iz istega pomnilnika. Ločena pomnilnika tudi omogočata, da je ukaz 14 biten, podatek pa 8 biten. Dostava ukaza je

v enem urinem ciklu, dvostopenjski tekoči trak (cevovod, pipe-line) pa omogoča prekrivanje dostavnega in izvrševalnega cikla. Zato rečemo, da se vsi ukazi odvijajo v enem ciklu, razen pogojnih skokov.

Mikrokontroler lahko naslavlja 1k 14-bitnih besed programskega pomnilnika, ki se nahaja na samem vezju. Programski ukazi se izvajajo samo znotraj vezja. PIC16C71 lahko naslavlja 48 registrov v podatkovnem nizu (register file). Vsi programsko dostopni registri so v podatkovnem pomnilniku.

4.7.3.1. O organizaciji programskega pomnilnika

Mikrokontroler PIC16C71 ima 13 bitni programski števec, z njim lahko naslavlja do 8k 14 bitnih besed. Samo prvih 1k * 14 bitov lokacij je v vezju samem, ostale pa je potrebno dodati zunaj. Reset vektor je kar naslov 0000H (PC ¬ 0000H) in prekinitveni vektor pa je na naslovu 00004H (PC ¬ M(0004H)). Ukazom CALL, RETLW, RETFIE, RETURN, ki potrebujejo sklad, je namenjenih 8 13-bitnih lokacij v posebnem hardverskem skladu, ki dela po načelu prvi_noter_zadnji_ven (FILO)!

4.7.3.2. Programski števec (PC)

PC je trinajst bitni register, ki ima spodnjih 8-bitov v r/w načinu, medtem ko zgornji del bajta (5 bitov) spremeni le ob nekaterih ukazih, preko registra PCLATH (program counter latch high), le-ti so:

- ukazi, ki so rezultat aritmetičnih operacij in spreminjajo PCL register, v PCH del pa se naloži vrednost iz PCLATH<4:0>,
- ukazi GOTO ali CALL dajo vrednost v PC<10:0> iz operandnega dela ukaza, v PC<12:11> pa se naložita bita iz PCLATH<4:3>. (Naslov PCLATH v registerskem nizu je 0Ah).

4.7.3.3. Sklad

Sklad je velikosti 8 13-bitnih besed. Le ta ni del niti pomnilnika, niti registerskega niza. Prav tako kazalec sklada (skladovni register) ni moč programsko spreminjati. Vrednost se v sklad naloži (push) z ukazom CALL ali ob prekinitvenem signalu. Iz sklada (pop) se vrednost pobere ob ukazih za vračanje: RETURN, RETLW (return lit. to reg W), RETFIE (return from interrupt). Na register PCLATH ne vpliva zapisovanje ali branje s sklada.

4.7.3.4. Registrski niz

Sestavljajo ga 8-bitni registri, ki so organizirani kot pomnilnik RAM velikosti 128*8 bitov. Dostop do lokacije je posreden ali neposreden, kar določa register FSR (file select register). Niz je razdeljen na dve strani (pages) registrov, od naslova 0 do 7Fh je 0. stran, od 80h do FFh je 1. stran. (Sl. 4.7.3.)

V registerskem nizu so naslednje lokacije:

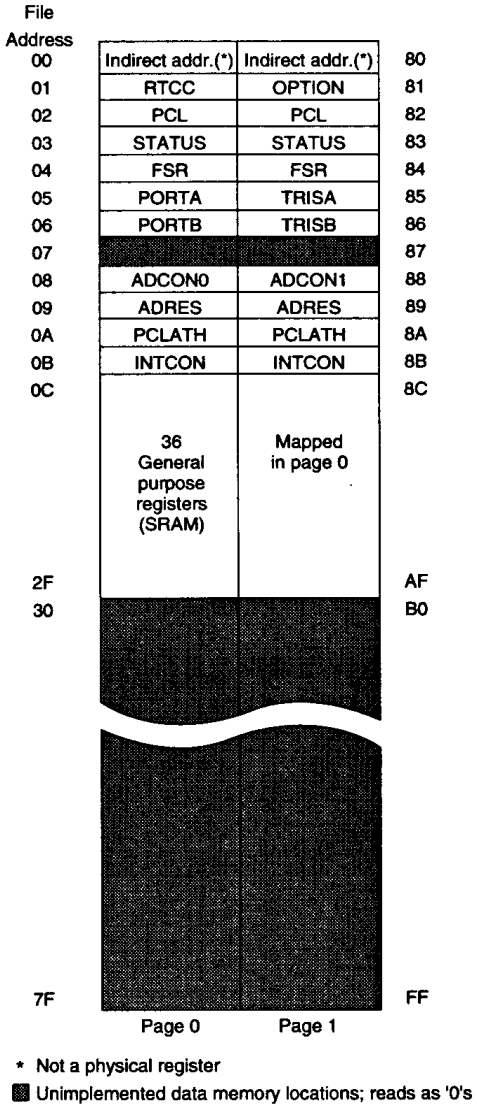
- od 0h do 08h, le-teh 12 lokacij hrani specialne funkcijske registre,
- od 0Ch do 2Fh, ki hranijo splošne registre, le-ti so realizirani kot statični pomnilnik RAM,
- od 80h do 8Bh, kamor se preslikajo nekateri registri iz niza 0h do 0Bh,
- od 8Ch do AFh so naslovi istih registrov kot 0Ch do 2Fh,

- od 30h do 7Fh in od B0h do FFh, ki so neuporabljeni spisek naslovov.

Naslavljanje registerskega niza poteka na več načinov:

- neposredni način > 9 bitov neposrednega naslova dobimo z združitvijo (konkatenacijo) 7 bitov iz operacijskega dela ukaza in z bitoma iz registra stanj (RP1, RP0). Le-ta - najtežja - bita povesta stran v registrskem nizu,

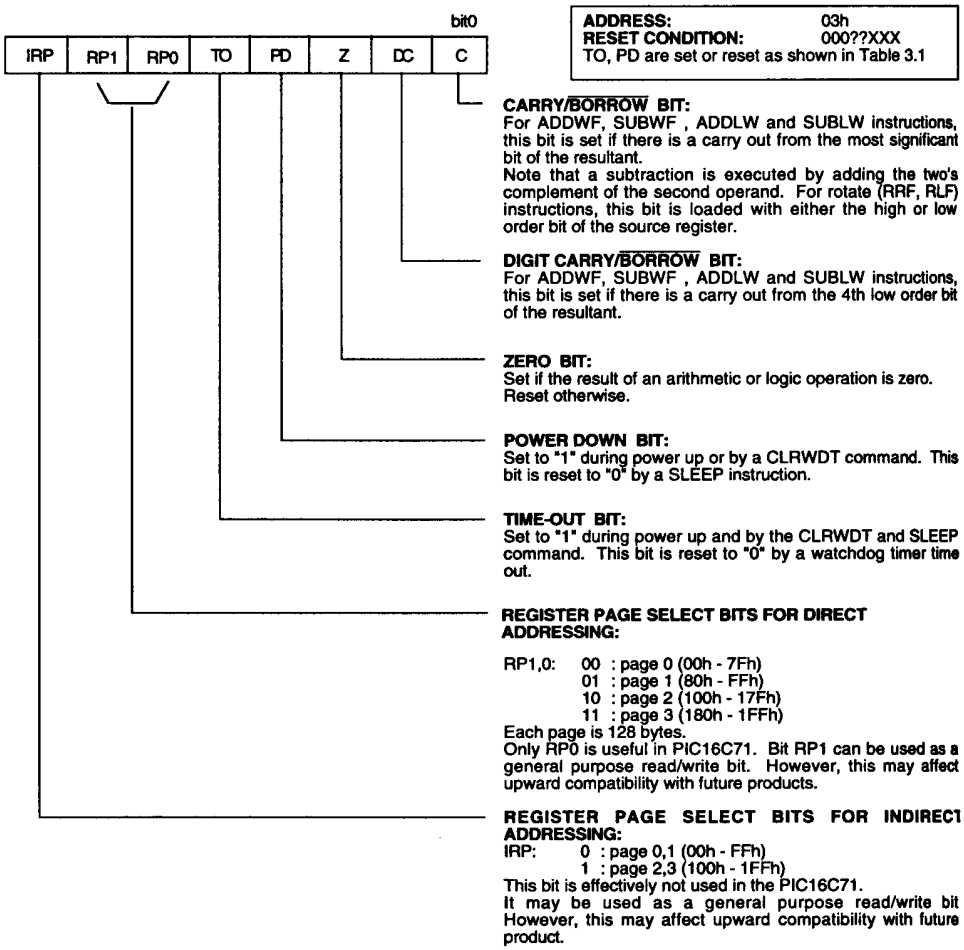
- posredni način > je možen z upoštevanjem naslova 00h v registerskem nizu, to je register za indirektno naslavljanje (INDF). Posredni naslov - 9 bitov - torej dobimo s sestavljanjem 8 bitov vsebine registra FSR in enim bitom IRP iz registra stanj. Najtežji bit iz registra FSR in bit IRP določata stran v registrskem nizu.



Slika 4.7.3. Registerski niz , strani 0. in 1., medtem ko strani 2. in 3. nista uporabljeni.

4.7.3.5. Register stanj

Tudi ta je register niza kot vsi drugi, s to razliko, da nekaterih bitov ni moč spreminjati, oziroma se spremenijo sami ob koncu ukaza (Sl. 4.7.4.).



Slika 4.7.4. Register stanj.

4.7.3.6. ALE

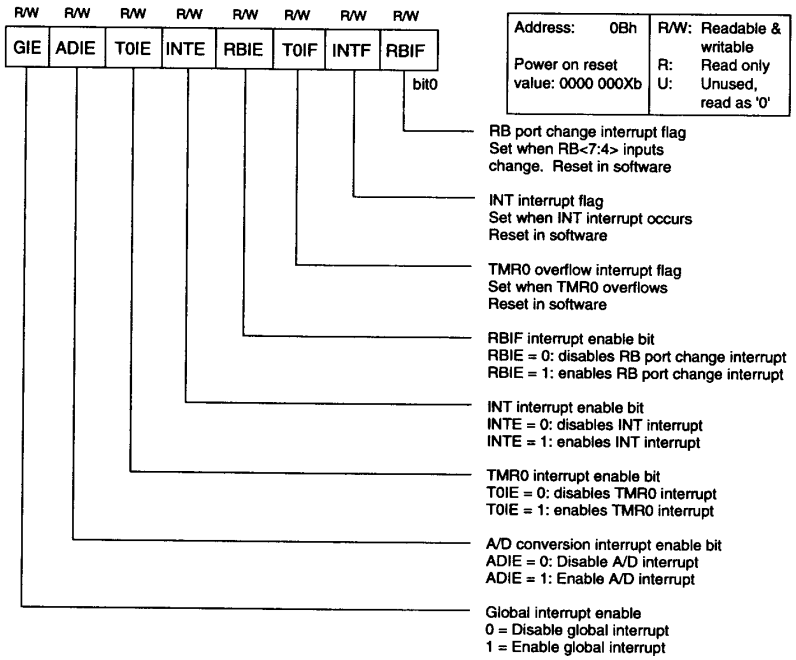
ALE je 8-bitna in obvlada seštevanje , odštevanje, pomikanje in logične operacije. Rezulatate sharanjuje v akumulatorju W. Le-ta register tudi ni mogoče naslavljati nepostredno, pač pa je njegov naslov vsebovan v ukazu.

4.7.3.7. Prekinitve

Imamo 4 izvore prekinitev:

- zunanji (INT) izvod,
- ob prelivu časovnika (timer TMR0),
- ob koncu A/D konverzije,
- sprememba na izvodih PB<7:4>.

Zahteve po prekinitvah se zabeležijo v registru INTCON (Sl. 4.7.5.).



Slika 4.7.5. INTCON register.

4.7.4. Nabor ukazov

Tabelo na Sliki 4.7.1. bomo razumeli, če povemo, kaj pomenijo naslednje črke:

polje	opis
f	naslov v nizu registrov (od 00h do 07Fh)
w	delovni register, akumulator
b	naslov bita znotraj 8-bitnega registra niza
k	znak, konstanta ali labela
x	neuporabljena vrednost, {0, 1}
d	ponorni register: d=0 -> shrani rezultat v W, sicer v register f
!T0	iztek časovne kontrole
!PD	izklop napajalne napetosti
DC	desetiški prenosni bit (po 4. bitu)
Z	vrednost je nič
C	prenosni bit (po 8. bitu)

Primer:

Napišimo primer programa za množenje dveh 8-bitnih vrednosti z zbirnikom za PIC:

```
; The 16 bit result is stored in 2 bytes
; Before calling the subroutine " mpy ", the multiplier should
; be loaded in location " mulplr ", and the multiplicand in
; " mulcnd " . The 16 bit result is stored in locations
; H_byte & L_byte.
```

Mnemonic, Operands		Description	Cycles	14-Bit Opcode msb lsb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS						
ADDWF	f, d	Add W and f	1	00 0111 dfff ffff	C, DC, Z	1, 2
ANDWF	f, d	AND W and f	1	00 0101 dfff ffff	Z	1, 2
CLRF	f	Clear f	1	00 0001 1fff ffff	Z	2
CLRW	-	Clear W	1	00 0001 0xxx xxxx	Z	
COMF	f, d	Complement f	1	00 1001 dfff ffff	Z	1, 2
DECF	f, d	Decrement f	1	00 0011 dfff ffff	Z	1, 2
DECFSZ	f, d	Decrement f, Skip if 0	1 (2)	00 1011 dfff ffff		1, 2, 3
INCF	f, d	Increment f	1	00 1010 dfff ffff	Z	1, 2
INCFSZ	f, d	Increment f, Skip if 0	1 (2)	00 1111 dfff ffff		1, 2, 3
IORWF	f, d	Inclusive OR W and f	1	00 0100 dfff ffff	Z	1, 2
MOVF	f, d	Move f	1	00 1000 dfff ffff	Z	1, 2
MOVWF	f	Move W to f	1	00 0000 1fff ffff		
NOP	-	No Operation	1	00 0000 0xx0 0000		
RLF	f, d	Rotate left f through carry	1	00 1101 dfff ffff	C	1, 2
RRF	f, d	Rotate right f through carry	1	00 1100 dfff ffff	C	1, 2
SUBWF	f, d	Subtract W from f	1	00 0010 dfff ffff	C, DC, Z	1, 2
SWAPF	f, d	Swap halves f	1	00 1110 dfff ffff		1, 2
XORWF	f, d	Exclusive OR W and f	1	00 0110 dfff ffff	Z	1, 2
BIT-ORIENTED FILE REGISTER OPERATIONS						
BCF	f, b	Bit Clear f	1	01 00bb bfff ffff		1, 2
BSF	f, b	Bit Set f	1	01 01bb bfff ffff		1, 2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01 10bb bfff ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01 11bb bfff ffff		3
LITERAL AND CONTROL OPERATIONS						
ADDLW	k	Add literal to W	1	11 111x kkkk kkkk	C, DC, Z	
ANDLW	k	AND literal to W	1	11 1001 kkkk kkkk	Z	
CALL	k	Call subroutine	2	10 0kkk kkkk kkkk		
CLRWDT	-	Clear watchdog timer	1	00 0000 0110 0100	$\overline{TO}$, $\overline{PD}$	
GOTO	k	Go to address	2	10 1kkk kkkk kkkk		
IORLW	k	Inclusive OR literal to W	1	11 1000 kkkk kkkk	Z	
MOVLW	k	Move literal to W	1	11 00xx kkkk kkkk		
RETFIE	-	Return from interrupt	2	00 0000 0000 1001		
RETLW	k	Return with literal in W	2	11 01xx kkkk kkkk		
RETURN	-	Return from subroutine	2	00 0000 0000 1000		
SLEEP	-	Go into standby mode	1	00 0000 0110 0011	$\overline{TO}$, $\overline{PD}$	
SUBLW	k	Subtract W from literal	1	11 110x kkkk kkkk	C, DC, Z	
XORLW	k	Excl. OR literal to W	1	11 1010 kkkk kkkk	Z	

Notes: 1. When an I/O register is modified as a function of itself (e.g. MOVF Port B,1), the value used will be that value present on the pins themselves. For example, if the data latch is "1" for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.
2. If this instruction is executed on the RTCC register (and, where applicable, d=1), the prescaler will be cleared if assigned to the RTCC.
3. If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a NOP.

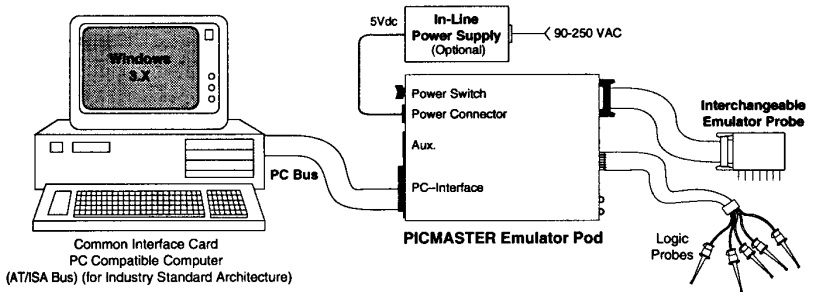
Slika 4.7.6. Ukazi pri PIC16C71 mikrokontrolerju.

```
;       Performance :
;                       Program Memory    :  15 locations
;                       # of cycles             :  71
;                       Scratch RAM             :   0 locations
;
;  This routine is optimized for code efficiency ( looped code )
;;*****************************************************************
;
            TITLE             "multiplier"
            LIST              P=16C71
mulcnd      equ     09        ; 8 bit multiplicand
mulplr      equ     10        ; 8 bit multiplier
H_byte      equ     12        ; High byte of the 16 bit result
L_byte      equ     13        ; Low byte of the 16 bit result
count       equ     14        ; loop counter
STATUS      equ     03        ; STATUS register F3
CARRY       equ     0         ; Carry bit in status register
Same        equ     1         ;
;
;
```

```
; ****************************          Begin Multiplier Routine
mpy_S       clrf        H_byte
            clrf        L_byte
            movlw       8
            movwf       count
            movf        mulcnd,w
;Clear the carry bit in the status Reg:
            bcf         STATUS,CARRY
loop        rrf         mulplr
            btfsc       STATUS,CARRY
            addwf       H_byte,Same
            rrf         H_byte,Same
            rrf         L_byte,Same
            decfsz      count
            goto        loop
            retlw       0
            END
```

4.7.5. Razvojna orodja

Z vezjem PIC proizvajalca Microchip je možno nabaviti dodatno programsko opremo, ki je uporabna na PC osebnih ali združljivih računalnikih. Dobimo jo pod imeni:



Slika 4.7.7. Razvojno orodje PICMASTER.

- PICMASTER na Sl. 4.7.7. (z emulatorjem),
- PIC MATE (z univerzalnim programatorjem),
- PIC START (ceneni programator),
- zbirnik za PIC
- programski simulator.

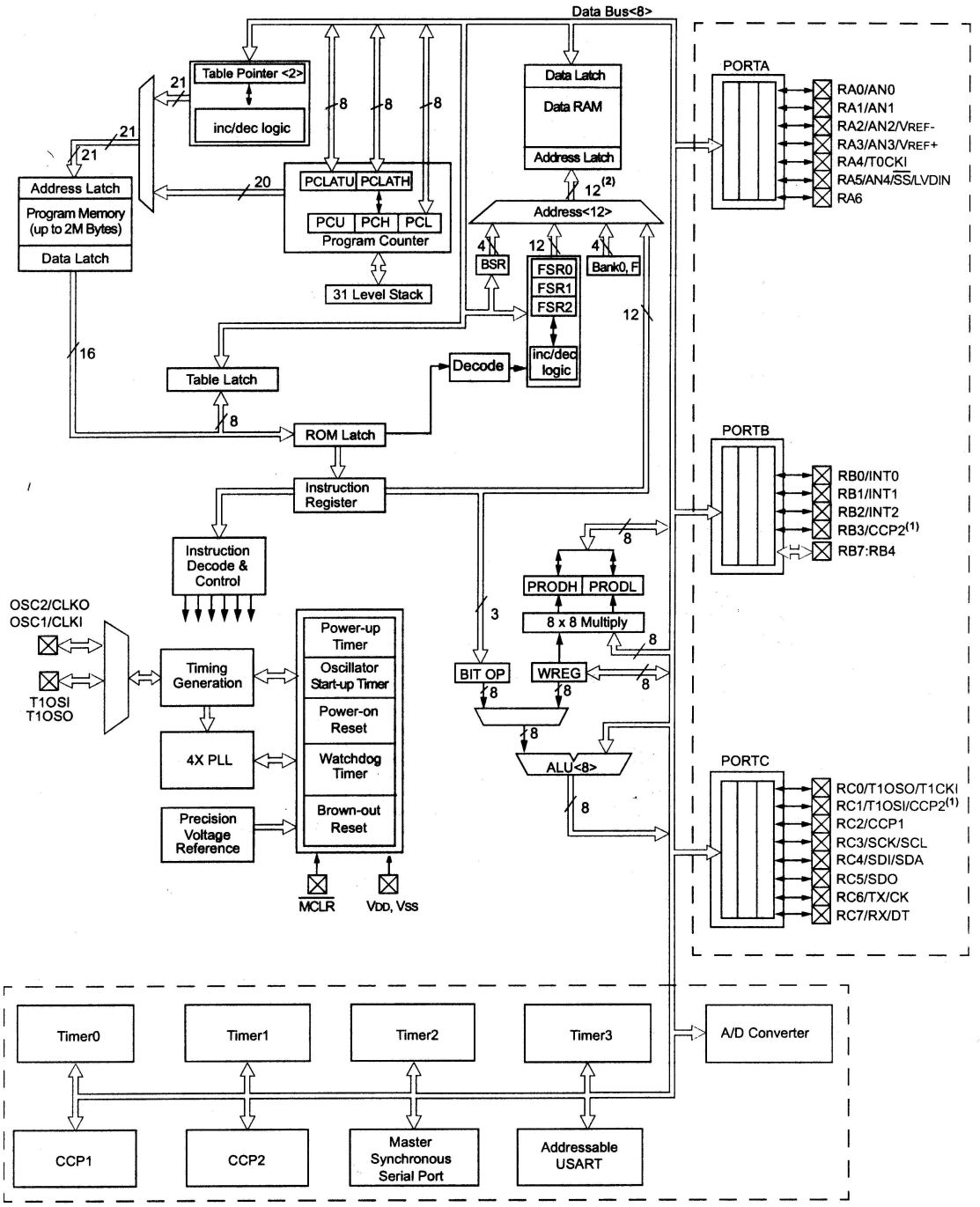
4.7.6. Mikrokontroler družine PIC 18C

Proizvajalec Microchip je leta 2001 razvil in ponudil tržišču zmogljivejše vezje družine 18. Njihove lastnosti beremo s Tab. 4.7.2.

Če družini 17 in 18 primerjamo med seboj, najprej ugotovimo, da vezji nista kompatibilni. Ostale prednosti družine 18 (Sl. 4.7.8.) pa so:

a. ukazi so 16-bitni,
b. naslovna vodila so 21-bitna,
c. podatkovni pomnilnik uporabljamo kot niz registrov, oziroma kot niz lokacij RAM. Registri so združeni v 16 bank, s po 256 registri /lokacijami/.

d. sklad ima 31 lokacij,
e. CPE deluje kot procesor v RISC arhitekturi, z dvostopenjskim tekočim trakom (dveh stanj: dostave in eksekucije),



Slika 4.7.8. Mikrokontroler družine 18Cxx2.

Tab. 4.7.2. Lastnosti mikrokontrolerjev Microchip družine 18.

opis	PIC18C242	PIC18C252	PIC18C442	PIC18C452
del. ura	40 Mhz	40 Mhz	40 Mhz	40 Mhz
progr. pom.	16k bajt	32k bajt	16k bajt	32k bajt
pod. pom	512	512 + 1024	512	512 + 1024
št. ukazov	75	75	75	75
prekinitev	16	16	17	17
I/O porti	A, B, C	A, B, C	A, B, C, D, E	A, B, C, D, E
časovniki	4	4	4	4
modul za zajem in prim.	2	2	2	2
serijske kom.	MSSP, USART	MSSP, USART	MSSP, USART	MSSP, USART
10 bit A/D	5 vhodov	5 vhodov	8 vhodov	8 vhodov
ohišja	28 DIP, 28 SOIC 28 JW	28 DIP, 28 SOIC 28 JW	40 DIP 44 PICC 44 TQFP 40 JW	40 DIP 44 PICC 44 TQFP 40 JW

f. CPE ima 8-bitni hardverski (matrični) pomnilnik,
g. Mikrokontrolerji se nahajajo v različnih ohišjih, od najmanjšega, ki je velikosti 0,5 x 0,5 cm, pa do 44 DIP.

Literatura

1) D. Kodek, Arhitektura računalniških sistemov, BiTIM, 2001;
2) David A. Paterson, J.L. Hennessy, Computer Organization & Design., The Hardware/software Interface, Morgan & Kaufmann, 1999;
3) Tom Shanley, Don Andreson, PCI System Architecture, Addison-Wesley Pub. Comp, 1995;
4) R.L. Sites, Alpha AXP Architecture, Com. of the ACM, Vol. 36, No. 2, pp. 33-44, 1993
5) Daniel C. Hyde, Using Verilog HDL to Teach Computer Architecture Concepts, *IEEE Comp. Society Technical Committee on Comp. Arch. Newsletter*, Feb. 1999, pp.31-33;
6) V. Guštin, Designing the Microprocessor with Abel-HDL, *Comp. Applications in Eng. Education*, Vol. 9, No. 2, pp. 87-92, 2001.
7) John R. Hauser, John Wawrzynek, Garp: A MIPS Processor with a reconfigurable Coprocessor, *The 5th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, April 16-18, 1997, pp. 12-21.
8) -, ALPHA Architecture Hand Book, Digital, 1992;
9) -, Alpha Architecture Reference Manual, 4 th Edition, ftp.compaq.com/pub/produtcts/alphaCPUdocs/alpha_arch_ref.pdf;
10) -, Intel Celeron processor, http://developer.intel.com/design/celeron/index.htm, 1999;
11) -, Intel Pentium III Xeon Procesor at 500 and 550 MHz, ftp://download.intel.nl/design/pentiumiii/xeon/datashts/24509402.pdf, 1999;
12) -; Microchip, PIC18CXX2, data Sheet, High Performance Microcontroler with 10-bit A/D, www.microchip.com
13) Walter A. Triebl, The 80386, 80486, and Pentium Processor, Hardware, Software, and Interfacing, Prentice Hall, 1996.