ARVUTID II

P R O T S E S S O R I D

Loengutsükkel

H. Mägi

2007

SISUKORD

1.1 Protsessori kontseptsioon.

Protsessori töö põhineb USA-s Princetoni Ülikoolis töötanud John von Neumanni poolt 1945. a. avaldatud mällu salvestatud programmiga arvuti ideel. Tema poolt pakutud arvutiarhitektuuri nimetatakse seepärast ka Princetoni arhitektuuriks, mis seisneb selles, et programm (s.o. käskude jada) koos töödeldavate andmetega salvestatakse arvuti mällu. Pärast käivitamist hakkab protsessor käskhaaval programmi täitma, kusjuures iga käsu täitmine koosneb järgmistest sammudest (joonis 1.1): :

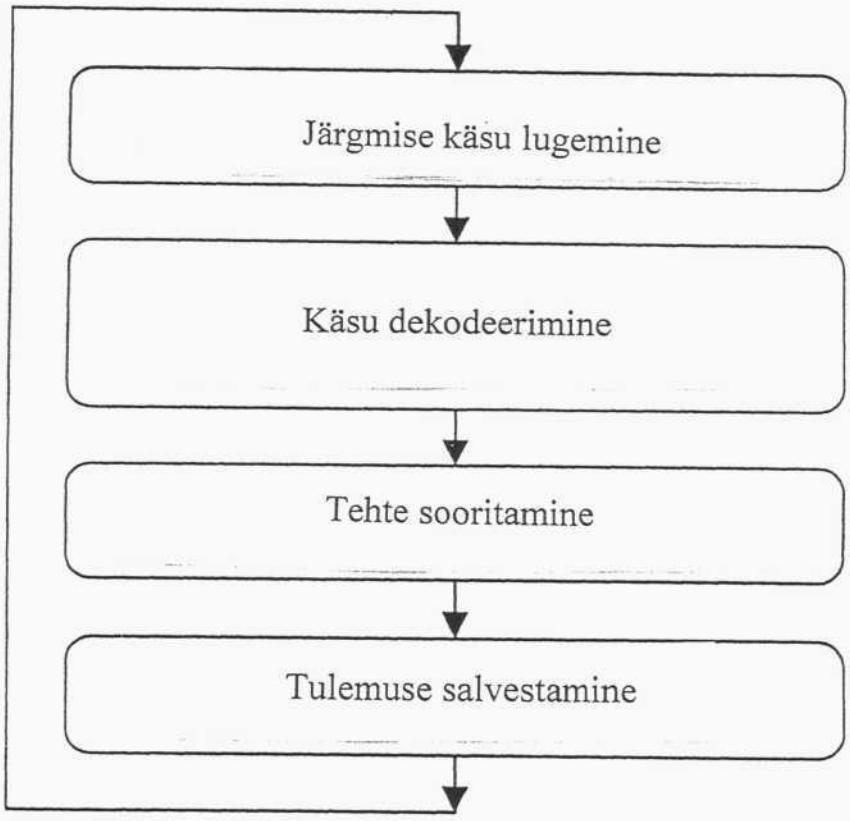
- järjekordse käsu lugemine mälupesast, mille aadress paikneb käsuloenduris. Loetud käsk fikseeritakse käsuregistris, kusjuures käsuloenduri sisu suurendatakse sõltuvalt käskude pikkusest, kas 1, 2 või 4 võrra, selleks et formeerida järgmise käsu aadress. Eelduseks on programmeerija poolt kirja pandud käskude paiknemine mälus pesade numeratsiooni järjekorras. Käskude täitmise järjekorra muutmiseks on ette nähtud siirdekäsud, mille abil muudetakse hüppeliselt käsuloenduri sisu ja seega käskude täitmise järjekord;
- käsu dekodeerimine jaguneb kahte etappi: käsukoodi dekodeerimine, mille tulemusena tuvastatakse sooritatav tehe ja operandide aadresside dekodeerimine, mille tulemusena tuvastatakse tehtes osalevad operandid;
- operandide lugemine mälust registritesse;
- tehte sooritamine aritmeetika-loogikaseadmes;
- tulemuse salvestamine registrisse;

Mällu salvestatuna võib programmi skemaatiliselt kujutada nii nagu joonisel 1.2, Käsud paiknevad pesades numeratsiooni järjekorras, nooltega varustatud siirdekäsud määravad käskude täitmise järjekorra. Käskude pikkus mõõdetuna baitides sõltub protsessori tüübist. Keeruka käsustikuga arvutite (CISC) protsessorid omavad väga erineva pikkusega käskusid, alates ühebaidistest ja lõpetades kuue ja enambaidiste käskudega. Vähendatud käsustikuga arvutite (RISC) protsessorite käsud on reeglina (väheste eranditega) ühepikkused ja käskude nomenklatuur on piiratud.

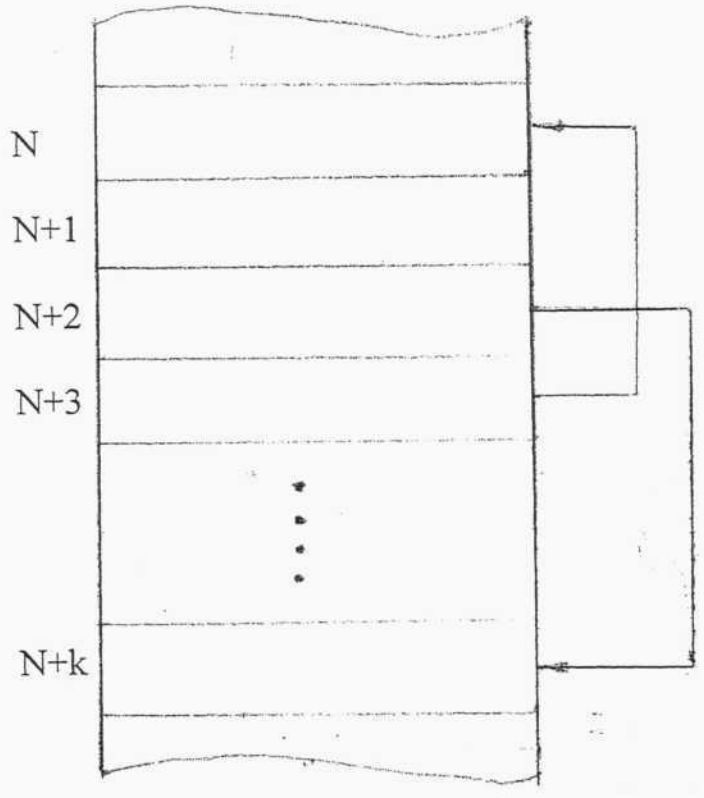
Protsessori poolt töödeldavate andmete formaat on mitmekesine. Täisarvulised operandid võivad olla ühebaidised (8 bitti), kahebaidised (16 bitti), neljabaidised (32 bitti) ja kaheksabaidised (64 bitti) või 18-kohalised kahendkodeeritud kümnendarvud (BCD). Naturaalarvulised e. ujukoma operandid omavad kahte formaati: lühike reaalarv (24 bitti) ja pikk reaalarv (53 bitti). Hüpoteetilise RISC tüüpi protsessori käskude ja andmete formaate on kujutatud joonisel 1.3.

Protsessori käsustik koosneb kolme tüüpi käskudest:

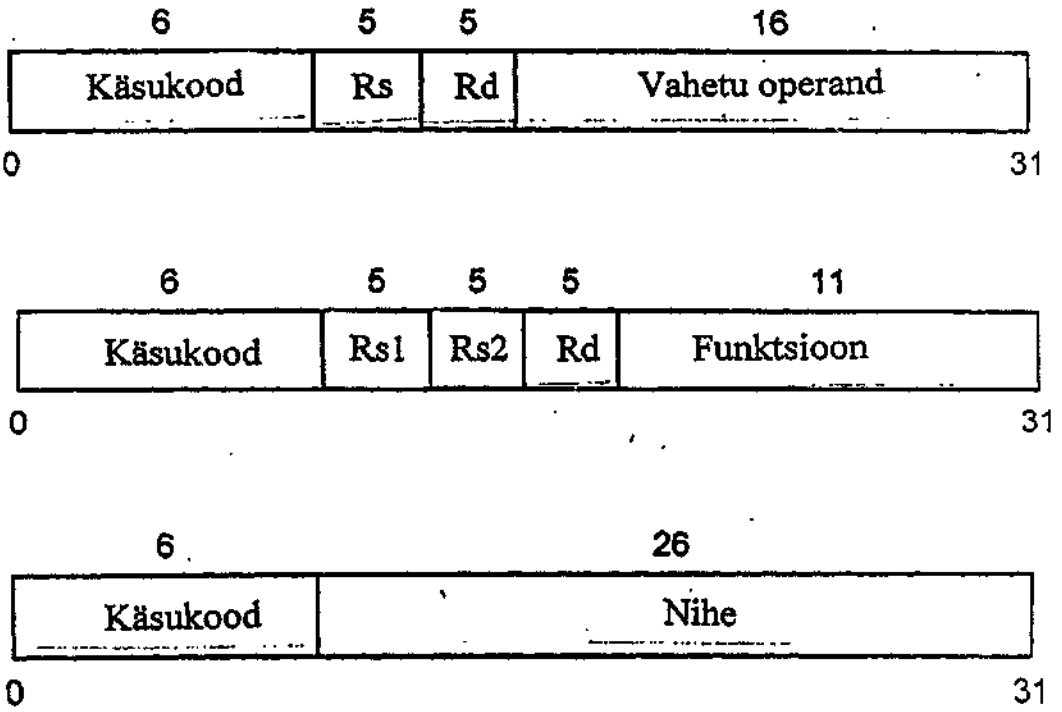
- andmeedastuskäsud siirdavad operande lähteregistrist (Rs) sihtregistrisse (Rd), vahetu operandi käsust sihtregistrisse (Rd), operande registrist mällu ja mälust registrisse;
- andmetöötluskäsud sooritavad aritmeetika-loogikatehteid;
- siirdekäsud muudavad käskude täitmise järjekorda, mis on määratud mälupesade numeratsiooniga;
-



Joonis 1. 1 Protsessori funktsionaalne algoritm



Joonis 1. 2 Mällu salvestatud programm



Joonis 1. 3 Käskude vormingud

Andmete vormingud	Piirkond	Täpsus	7 0 7 0 7 0 7 0 7 0 7 0 7 0 7 0 7 0 7 0
Täisarv	10^4	16 Bits	I_{15} I_0
Lühike täisarv	10^9	32 Bits	I_{31} I_0
Pikk täisarv	10^{18}	64 Bits	I_{63} I_0
Pakitud BCD	10^{18}	18 Digits	S — $D_{17}D_{16}$ D_1 D_0
Lühike reaalarv	$10^{\pm 38}$	24 Bits	S E_7 E_0 F_1 F_{23} F_0 peidetud
Pikk reaalarv	$10^{\pm 308}$	53 Bits	S E_{10} E_0 F_1 F_{52} F_0 peidetud

BCD (Binary Coded Decimal) – kahendkodeeritud kümnendarv

Kõik kolme tüüpi käsud on ühepikkused (32 bitti) ja sisaldavad kõik 6-bitiseid käsukoodi väljasid, võimaldades kodeerida kuni 64 erinevat käsumodifikatsiooni. Edastus- ja töötluskäskude 5-bitised aadressiväljad võimaldavad adresseerida 32-te protsessori registrit. Edastuskäskudes on 16-bitine vahetu operandi väli ja siirdekäskudes 26-bitine siirde sihtaadressi väli. Tehtekood koos funktsioonikoodiga andmetöötluskäsu 11-bitises funktsiooniväljas määrab käsuga sooritatava tehte.

Ülalkirjeldatud hüpoteetilise käsustikuga protsessori füüsiline struktuur (skeemikomponendid koos nendevaheliste andmesignaale kandvate ühendustega moodustavad protsessori riistvara) on kujutatud joonisel 1.4.Kõik protsessori registrid ja nendevahelised andmekanalid on 32-bitised. Juhtsignaalide kanaleid skeemil näidatud ei ole.

Skeemil on plokkidena kujutatud struktuuriühikud (punktiiriga piiritletud) , mis vastavad joonisel 1 toodud protsessori töö algoritmi sammudele: käsuvõtu plokk, käsu dekodeerimise plokk, tehte sooritamise plokk ja tulemuse salvestamise plokk.

Plokid on ühendatud järjestikku ja seega võib neid nimetada astmeteks, mida käsu täitmise protsessis sammhaaval läbivad töödeldavad andmesignaalid. Sammude täitmist sünkroniseerib protsessori taktigeneraator. Iga järgmise sammu täitmist saab alustada alles siis, kui eelmine on täidetud. Protsessori töö kiiruse määrab käsu täitmiseks kuluv aeg, mis võrdub selleks vajalike sammude sooritamise summaarse ajaga:

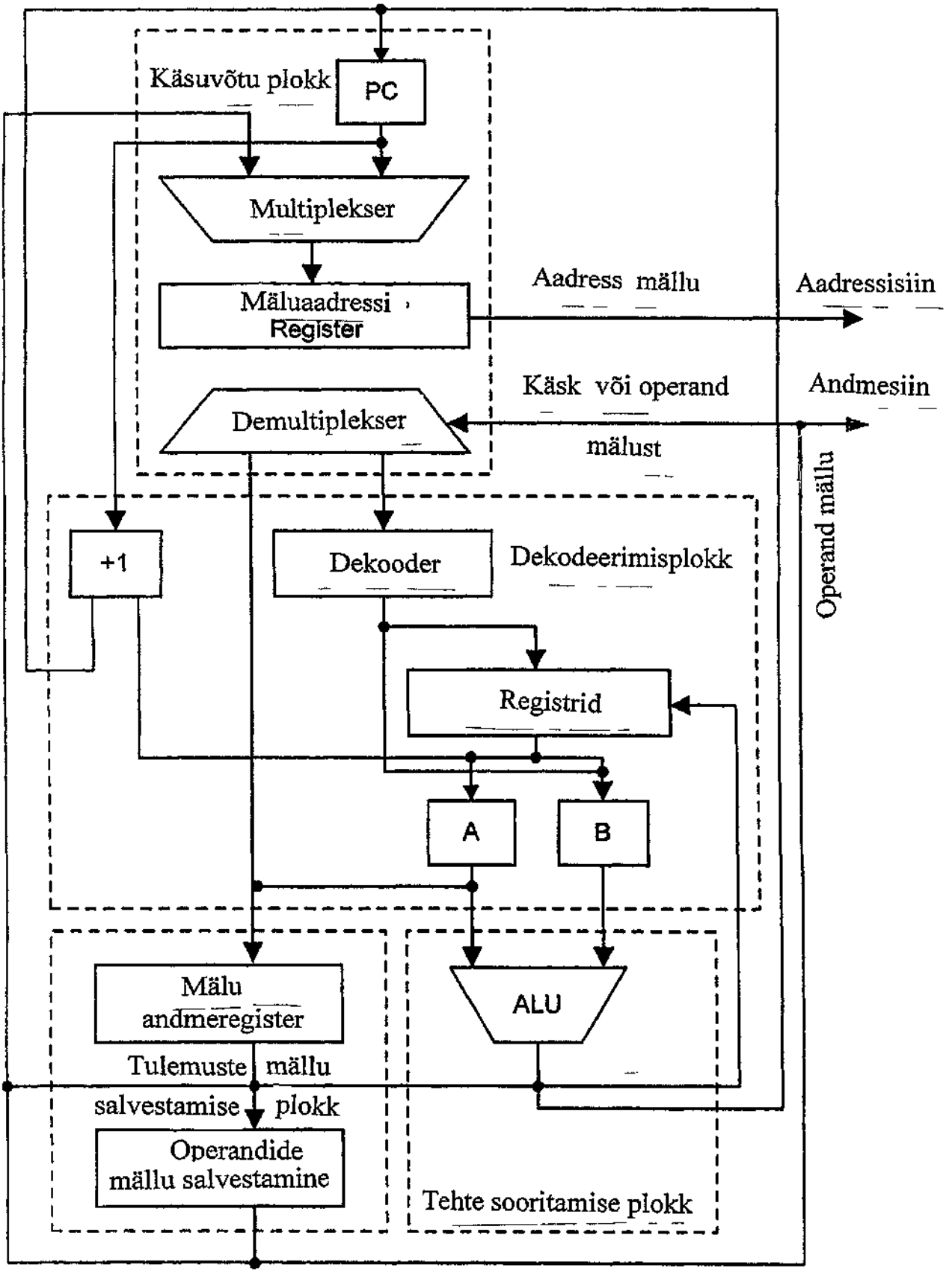
$$T = t_{kv} + t_{kd} + t_{ts} + t_{s}$$

T - käsu täitmise periood
t_{kv} käsuvõtu periood
t_{kd} - käsu dekodeerimise periood
t_{ts} - tehte sooritamise periood
t_{s} - tulemuse salvestamise periood

Protsessori üksikud astmed on erineva keerukusega, sisaldades erineva arvu skeemikomponente, mida signaalid järjestikku läbivad. Seega on erinevate sammude sooritamise aeg erinev. Käsu täitmise aeg sõltub ka sellest, millist käsku parajasti täidetakse. CISC – tüüpi protsessori käsustik koosneb suurest arvust erineva pikkuse ja suurtes piirides erineva täitmisajaga käskudest. RISC-tüüpi protsessori käsustik seevastu koosneb piiratud arvust erinevatest käskudest, mis on reeglina ühepikkused ja nende täitmise aja erinevused ei ole suured.

Käsu täitmise sammude erineva kestuse tõttu, tuleb käsu sammude sünkroniseerimise periood valida lähtuvalt maksimaalse kestusega sammust.

Von Neumanni printsiibist lähtudes täidab protsessor üksteisele järgnevad käsud puhtal kujul järjetikku: järgmise käsu täitmist ei alusta enne , kui eelmise käsu



Joonis 1. 4 Protsessori struktuur

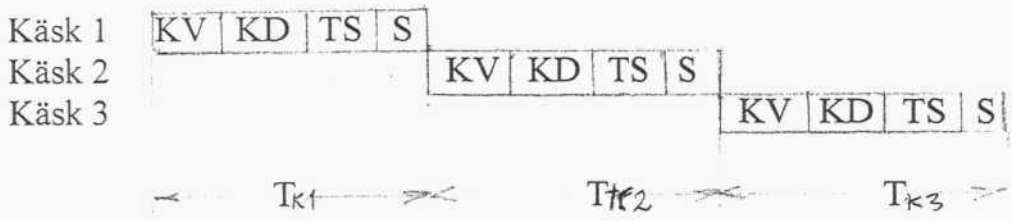
täitmine on lõpetatud (tulemus on salvestatud) (joonis 1.5). Käsu täitmise ajal on hõivatud ainult üks neljast protsessori astmest, mis viitab sellele et protsessori aparatuuri (riistvara) kasutatakse ainult ühe neljandiku ulatuses. Käsu täitmiseks kulub neli protsessori taktiperioodi: iga järgneva käsu täitmise tulemuse saame neljandal taktil.

1.2 Käskude konveiertöötlus.

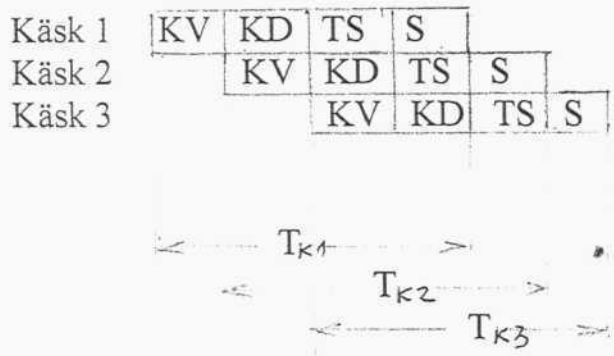
Olemasolevat aparatuurset reservi saab ära kasutada sel teel, et protsessori astmed pannakse tööle konveierina nii, nagu näidatud joonisel 1.6. Esimese käsuvõtuga mälust fikseeritakse käsk astmes 1 (takt t1). Vahetult pärast esimese käsu edastamist dekooderi astmesse 2, sooritatakse järgmine käsuvõtt astmesse 1 (takt t2). Taktil 3 edastatakse dekodeeritud käsk tehte sooritamise astmesse 3. Samal ajal edastatakse käsk 2 dekooderi astmesse ja toimub järgmine käsuvõtt, mille tulemusena fikseeritakse kolmas käsk astmes 1. Taktil 4 toimub neljas käsuvõtt, osaliselt täidetud käsud nihkuvad astme võrra edasi ning neljandas astmes fikseeritakse esimese käsu täitmise tulemus, mis mällu või registrisse salvestatakse. Alates neljandast taktist on konveieri kõik astmed tööga koormatud ja igal taktil fikseeritakse astmes 4 järjekordne tulemus jne.

Kui käskude täitmise puhul tavalise järjestikuse von Neumanni mudeli järgi väljastab protsessor igal neljandal taktil järjekordse tulemuse, siis konveieri rakendamisel juba igal taktil. Selline neljaastmelise konveieri puhul saavutatud neljakordne protsessori jõudluse suurenemine oleks võimalik vaid ideaaljuhul, kui käsustiku kõigi käskude täitmise vastavad sammud oleksid võrdse kestusega. Kahjuks pole seda võimalik saavutada, kuna erinevate käskudega sooritatavate tehete keerukus on erinev ja nende täitmine hõlmab erineva osa protsessori aparatuurist ja seega on signaalide viivitus nende täitmisel erinev. Konveieri sammu taktiperiood peab olema valitud vähemalt võrdseks kõige kestvama sammuga. Kõigi väiksema kestusega sammude sooritamisel konveieri vastavad astmed peatuvad enne taktiperioodi lõppu jäädes ootama järgmise taktiperioodi algust. Astmete väljundsignaalid tuleb ooteajaks salvestada, mis nõuab astmete väljundile rööpsisendi- ja rööpväljundiga registrite lisamist. See omakorda suurendab riistvara mahtu ja signaalide levimise viivitust.

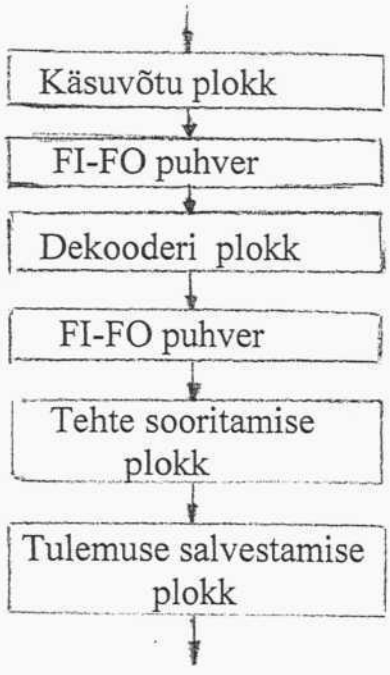
Mainitud põhjused ei piira olulisel määral konveieri rakendamisest tulenevat protsessori jõudluse tõusu. RISC- protsessorites, tänu piiratu käskude arvule ja lihtsustatud käskude vormingule, on konveiertöötluse rakendamine lihtsam ja annab paremaid tulemusi, kui CISC- protsessorites, kus suure arvu keeruka vorminguga käskude tõttu tuleb selleks rohkem täiendavat riistvara lisada. Näiteks, konveieri astmetevaheliste registrite asemel tuleb kasutada FI-FO tüüpi puhvreid, mis mahutaksid mitut eelmise astme poolt väljastatud tulemust. Kui tehte sooritamise plokis on käsil keeruka tehte sooritamine, mis vältab mitu takti ja ei hõiva andmesiini , saab käsuvõtu plokk lugeda mälust ja paigutada oma väljundpuhvrisse mitu järgnevat käsku. Samal ajal saab dekooderi plokk neid käske dekodeerida ja paigutada oma väljundpuhvrisse mitme dekodeeritud käsu täitmise signaalid. Kuna CISC- protsessoris on vähe registreid ja mällu



Joonis1. 5 Käskude järjestikune täitmine



Joonis1. 6 Käskude täitmine konveieril



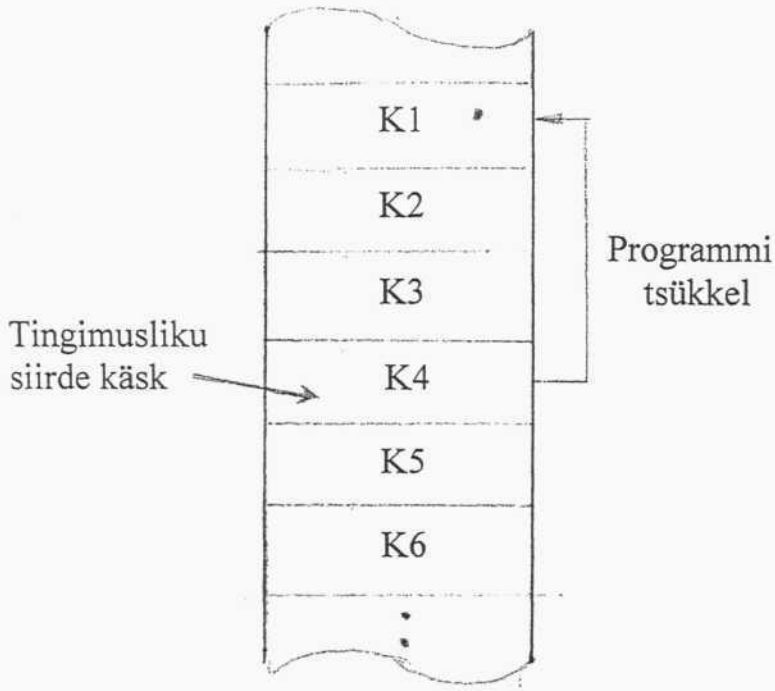
Joonis 1. 7

salvestamine on suhteliselt aeglane, siis tehte sooritamise ploki väljundis vajalik puhverregister, mis kuulub tulemuse mällu salvestamise ploki koosseisu. Puhvritega varustatud protsessori struktuur on toodud joonisel 1.7.
Käskude konveiertöötluse tõhusust vähendavad ka programmides paratamatult esinevad siirde- ehk üleminekukäsud. Siirdekäsud muudavad hüppeliselt programmide järjestikust, mälupesade numeratsiooni järgivat täitmist, suunates protsessori täitma programmilõiku, mis algab siirdekäsus näidatud aadressil. Siirdekäsud jagunevad tingimusteta ja tingimuslikeks. Eriti olulised on tingimuslikud siirdekäsud, mis võimaldavad programmeerida neis käskudes püstitatud tingimuste põhjal vahepealsetest tulemustest sõltuvaid hargnevusi ja kordamisi (tsükleid) arvutusprotsessides. Teiste sõnadega, tingimuslike siirdekäskude abil omistatakse programmidele loogiliste otsuste tegemise võime, milleta programmeerimine oleks võimatu.
Tingimusliku siirdekäsu täitmisel valib protsessor ühe kahest võimalikust käskude täitmise
suunast: käsukoodiga määratud tingimuse mittetäitmise puhul jätkab programmi täitmist siirdekäsule järgneva käsuga, tingimuse täitmise puhul aga siirdub täitma programmi, mille algaadress on toodud siirdekäsu aadressiväljas. Tingimuse täitmine sõltub siirdekäsule eelneva käsu täitmise tulemusest, mis võib olla null (0), positiivne (+) või negatiivne (--). Tulemuse tunnust, mis on fikseeritud protsessori tunnuste (lippude) registris vastava bitiga, võrreldakse käsuga määratud tingimusega. Tunnuse ja tingimuse ühtimisel siire toimub, vastasel juhul mitte. Näiteks, kui siire peab toimuma nullise tulemuse puhul s.o. täidetakse käsku JZ (Jump Zero)) ja tegelik tulemus on null, s.o. vastav lipp on olekus 1 (Z=1), siire toimubki. Vastasel juhul, kui tulemus on mitte null s.o. Z=0, siiret ei toimu.

Kuidas mõjutab tingimuslik siirdekäsk konveieri tööd? Seda on kujutatud tsükkelprogrammi näitel joonisel 1.8. Kui protsessor võtab mälust järjekordse käsuna tingimusliku siirdekäsu K4, siis tuvastab ta selle alles dekodeerimise sammul konveieri teises astmes. Vahepeal on esimesse astmesse võetud juba siirdekäsule järgnev käsk K5. Järgmisel konveieri töötaktil liigub siirdekäsk K4 tehte sooritamise astmesse, kus võrdlustehte tulemusena selgub, kas siire tegelikult toimub, või mitte. Samal ajal liigub käsk K5 dekodeerimisastmesse ja käsuvõtu astmesse saabub käsk K6. Võrdlustehte tulemus võib olla kas siiret välistav või kinnitav. Siiret välistaval juhul jätkab protsessor programmi täitmist juba alustatud suunas K5, K6, jne. Kui aga tulemus on kinnitav, siirdub protsessor täitma programmilõiku, mis algab käsuga K1 pärast pooleldi täidetud käskude K5 ja K6 tühistamist. Konveieri töö neljale tulemuslikule taktile järgnevad kaks tühja takti t8 ja t9, mis tulemust ei anna, mistõttu protsessori jõudlus väheneb kolmandiku võrra.
Tsükleid on programmides palju ja neid täidetakse tavaliselt väga suur arv kordi. Seepärast on väga oluline tingimussiirde käskudest tingitud konveieri jõudluse langust leevendada. Selleks kasutatakse mitmesuguseid meetodeid programmi hargnemise ennustamiseks (branch prediction). Tingimussiirde käsk on sisuliselt

	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10
Aste 1	K1v	K2v	K3v	K4v	K5v	K6v	K1v	K2v	K3v	K4v
Aste 2		K1d	K2d	K3d	K4d	K5d		K1d	K2d	K3d
Aste 3			K1ts	K2ts	K3ts	K4ts			K1ts	K2ts
Aste 4				K1s	K2s	K3s	K4s			K1s

Tühjad taktid



Joonis 1. 8 Tingimusliku siirdekäsu mõju konveieri tööle

	t1	t2	t3	t4	t5	t6	t7	t8	t9
Aste 1	K1v	K2v	K3v	K4v	K5v	K1v	K2v	K3v	K4v
Aste 2		K1d	K2d	K3d	K4d		K1d	K2d	K3d
Aste 3			K1ts	K2ts	K3ts	K4ts		K1ts	K2ts
Aste 4				K1s	K2s	K3s	K4s		K1s

Tühi takt

Joonis 1.9 Lihtsa hargnemise ennustamise effekt

programmi kahte suunda hargnemise punkt, mille puhul on võimatu üheselt kindlaks määrata, millise suuna programm valib, enne kui käsk ei ole dekodeeritud ja tingimus kontrollitud. Seepärast tuleb võtta appi ennustamine. Lähtuda tuleb seejuures oletusest, mille täideminekuks on teatav tõenäosus. Näiteks, programmitsüklite puhul , mida täidetakse suur arv kordi, on tõenäoline, et pärast eksimust ennetaval haru valikul esimesel tsüklil, saab seda kõikidel järgnevatel tsüklitel vältida, kui see eksimus konveieri juhtimisskeemis fikseerida ühe trigeri oleku näol, mis lubab teostada iga järgneva siirde ennetavalt, tingimuse täitmist kontrollimata. Iga programmitsükli täitmisel tekib seejuures konveieril ainult üks tühi takt (joonis 1.9). Programmi viimase tsükli täitmisel muutub protsessori lipuregistris asuva tsüklit juhtiva lipu olek keelavaks, mis suunab siirdekäsu selle täitmise kolmandal sammul tingimuse täitmist kontrollima, mille tulemusena selgub tsükli tingimuse mittetäitmine ja protsessor siirdub täitma siirdekäsule järgnevat käsku (s.o. väljub tsüklist), tekitades seega kaks tühja takti.Kirjeldatud lihtne hargnemise ennustamise võte võimaldab tsükkelprogrammide täitmisel tingimussiirde käskudest põhjustatud konveieri jõudluse langust vähendada ligemale kaks korda. Kasutades keerukamaid ennustusmeetodeid ja lisades nende realiseerimiseks vajalikku täiendavat riistvara, saab jõudluse langust praktiliselt vältida.

Konveieri jõudlust mõjutab ka üksteisele järgnevate käskude omavaheline andmesõltuvus, mida saab tuvastada nende dekodeerimise sammul vastavate registrite aadresside võrdlemise teel, milleks peab olema vastav loogika. Lihtsam on andmesõltuvuse tekkimist vältida, kuna sellega tuleb edukalt toime programmi kompilaator. Eelduseks seejuures on, et protsessoril oleks piisav arv registreid.

Andmesõltuvuse olemust selgitab alljärgnev kolmest käsust koosnev programmilõik, mille täitmist konveieril illustreerib joonis 1.10.

```
K1: ADD  R1, R2, R3
K2: SUB  R3, R4, R6
K3: XOR  R1, R5, R3
```

Esimese käsu (K1) täitmise tulemus, mis tekib registris R3 on operandiks järgmisele käsule (K2). Käsu K1 täitmine lõpeb takti t3 lõpul ja tulemuse salvestamine registrisse R4 alles takti t4 lõpuks. Käsu K2 korrektseks täitmiseks aga on seda tulemust tarvis t4 alguseks. Kui konveieri juhtskeem tekkivat operandi hilinemist ei väldi, loeb käsk K2 registrist R3 vana sisu, mille tulemusena tekib viga. Et seda vältida, tuleb käsu K2 ja kõigi järgnevate käskude täitmine peatada takti t3 lõpul üheks taktiperioodiks, mistõttu väheneb konveieri jõudlus.

1.3 RISC versus CISC.

Protsessoreid hakati liigitama RISC ja CISC protsessoriteks möödunud sajandi 80-date alguses, kui mikroprotsessorid olid arvutitehnikas võitmas juhtpositsiooni. Nii RISC kui ka CISC tunnustega protsessoreid projekteeriti ja

Käskude andmesõltuvus

	T1	t2	t3	t4	t5	t6	t7
Aste 1	K1v	K2v	K3v	K3v	K4v	K5v	K6v
Aste 2		K1d	K2d	K2d	K3d	K4d	K5d
Aste 3			K1ts		K2ts	K3ts	K4ts
Aste 4				K1s	seisak	K2s	K3s

Joonis 1.10

K1: ADD R1, R2, R3

K2: SUB R3, R4, R6

K3: XOR R1, R5, R3

toodeti juba suurte arvutite ja sellele järgneval miniarvutite ajastul. Esimeste mikroprotsessorite projekteerimisel ja nende baasil mikroarvutite ehitamisel 70-datel, võeti malli eelnevate põlvkondade arhitektuurist ja kohandati seda nii, et protsessor tervikuna mahuks ühele kristallile. Oli käibel isegi termin ühekristalne mikroprotsessor versus mitmekristalne- e. silpprotsessor, mille aritmeetikaseade pandi kokku mitmest erineval kristallil valmistatud protsessori silbist, igaüks 2- või 4-järguline.

Peamine rõhk oli asetatud siiski ühekristalsete protsessorite arendamisele. Tolle aja pooljuhttehnoloogia võimalused olid piiratud, mistõttu kristallile sai paigutada vaid minimaalne protsessori funktsioneerimiseks vajalik skeemikomponentide hulk. Registrite arvu tuli piirata, aparatuurse realisatsiooni asemel tuli käskude täitmise juhtimine üles ehitada püsimällu salvestatud mikroprogrammidel.

Mälukiipide väikese mahu ja kõrge hinna tõttu, oli operatiivmälu (põhimälu) piiratud mahuga. Seetõttu püüdsid konstruktorid koostada mikroprotsessorile sellise käsustiku , et iga käsk sisaldaks võimalikult palju tehte sooritamiseks vajalikku informatsiooni. See omakorda nõudis palju erinevaid mälu adresseerimisviise. Käsud said keerulise formaadiga ja erineva pikkusega, kuid see-eest nendest koostatud programmid olid kompaktsed ja hõivasid vähe mälu. Lisaks on CISC protsessorid kergesti programmeeritavad assembleris (iseg käsitsi).

Mainitud objektiivsetel põhjustel valmisidki CISC protsessorid, milliste esmaseks loojaks ja jõuliseks turuletoojaks oli 1978.a. USA firma **Intel** mudeliga **8086.** Intel suutis väga lühikese ajaga organiseerida 8086 massitootmise ja toodangu turule paiskamise. Protsessori vastu valitses suur huvi nii akadeemilistes , kui äri- ja tööstusringkondades. Arvutitööstuse gigant Firma IBM valis oma esimese personaalarvuti IBM PC protsessoriks 8086 8-bitise andmesiiniga versiooni **8088.** Ainult 8 üldregistriga, suurest arvust (üle 200) erineva pikkuse ja keeruka formaadiga käskudest koosneva käsustikuga ja 10 erineva adresseerimisviisiga on 8086 tüüpiline CISC protsessor. IBM PC võitis turul erakordse populaarsuse, mistõttu Intel'il , kui IBM-le protsessorite tarnijal avanesid avarad võimalused mikroprotsessorite arendamiseks ja tootmiseks. Üksteise järel ilmusid turule mudelid 80286 (milest sai IBM PC AT protsessor), 80386, i480, Pentium, Pentium Pro, Pentium II, Pentium III ja Pentium 4.

Kohe pärast 8086 turule tulekut hakati selle jaoks hoolega kirjutama tarkvara teaduse , tehnika , majanduse jne. jaoks. Tarkvara maksumus on paju kordi suurem protsessori ja selle baasil ehitatud arvuti riistvara maksumusest. Tarkvara,mille maht oli kiiresti kasvanud aukartustäratavaks, miljarditesse dollaritesse ulatuva maksumusega, tugineb otseselt protsessori CISC-tüüpi käsustikule , millest loobumine ei tulnud enam kõne allagi. Käsustiku muutmine oleks tähendanud kogu juba olemasoleva tarkvara tühistamist, mida selle laialdase leviku ja kõrge maksumuse tõttu lubada ei saanud. Seepärast tuli iga järgmise protsessori mudeli konstrueerimisel tagada tema ühilduvus eelmiste

jaoks kirjutatud tarkvaraga, mis tähendab, et kõik eelmiste mudelite jaoks kirjutatud programmid peavad töötama järgmistel mudelitel. Iga järgmine moodsam mudel erineb reeglina eelmisest suurema jõudluse, uute käskude lisamisega laiendatud käsustiku ja sellest tulenevalt suuremate võimalustega.

RISC protsessori peamiseks erinevuseks on selle käsustik, mis koosneb piiratud arvust fikseeritud pikkusega ja lihtsa formaadiga käskudest, milles kasutatavate adresseerimisviiside arv on väike. Käskude lihtne funktsionaalsus võimaldab kompileerimisel programme ka lihtsamini optimeerida. RISC arhitektuur on projekteeritud silmas pidades käskude konveiertöötlust, mistõttu konveier töötab effektiivsemalt kui CISC protsessoris.
RISC arhitektuuri lühiiseloomustus võiks olla järgmine:

- Fikseeritud pikkusega käsud lihtsustavad käsuvõttu.
- Väike arv erinevaid käsuformaate lihtsustab käskude dekodeerimist.
- Käsustiku laadimisel/ salvestusel põhinev ülesehitus võimaldab eraldada mällupöördumised arvutustest ja neid üksteisest sõltumatult optimeerida.
- Käskude lihtne funktsionaalsus lubab lihtsustada juhtseadet. Aparatuurselt realiseeritud juhtseade töötab kiiremini kui mikroprogramne
- Konveiertöötlusele orienteeritud arhitektuur optimeerib käskude täitmise kiiruse.
- Adresseerimisviiside väike arv kiirendab protsessori tööd.
- Rõhku on pandud kompaileri funktsioonide optimeerimisele, kuna arhitektuur on projekteeritud mitte assembleri, vaid kõrgtaseme keelte toetamiseks.
- Keerukus on kätketud kompailerisse (s.o. tarkvarasse), mitte protsessori riistvarasse.
- Kolmeoperandilised käsud kergendavad kompaileril optimeerida programme.
- Väiksemahuline aparatuurne juhtseade jätab piisavalt ruumi suurele arvule registritele (32 ja enam registrit), mis on vajalikud kompailerile programmide optimeerimiseks.

2.1 Arvuti mälusüsteem.

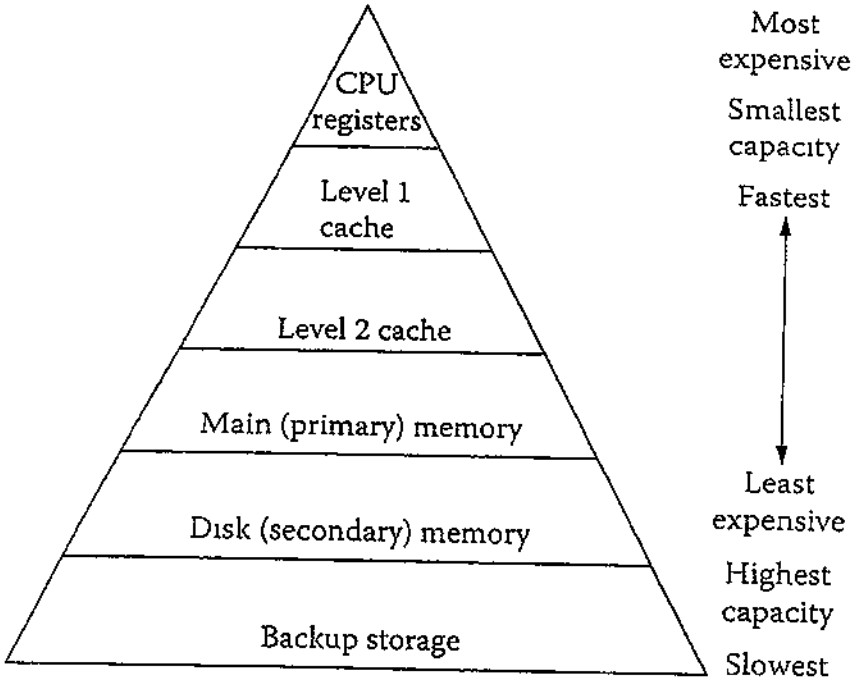
Kaks peamist arvuti mälule esitatavat nõuet on võimalikult suur andmemaht ja seejuures ka võimalikult suur kiirus (võimalikult väike reaktsiooniaeg pöördumisel). Kahjuks ei ole need nõuded seni ühelgi teadaoleval füüsikalisel põhimõttel ehitatud mäluseadmes täidetavad. Mälu andmemahu suurendamisel väheneb paratamatult kiirus, kuna seejuures suurenevad mäluploki füüsilised mõõtmed ja omakorda sellest tingitult pikenevad elektriahelad ja neis signaalide levimise viivitus. Siit järeldub, et viivituse vähendamiseks s. o. kiiruse suurendamiseks tuleb mäluploki andmemahtu vähendada.

Siit järeldub, et mingil ühel põhimõttel töötava monoliitse mäluplokiga ei ole võimalik arvuti mälu dilemmat lahendada. Mälu tuleb koostada hierarhilisena vähemalt kahest või enamast erineva kiiruse, mahu ja tööpõhimõttega seadmest. Kaasaegse arvuti hierarhilist mälusüsteemi võib kujutada püramiidina (joonis 2.1), mille tipus, s.o. kõige lähemal protsessorile asub maksimaalse kiirusega seade, mille maht on väga piiratud ja selle moodustavad protsessori registrid. Püramiidi aluse moodustab aga seade, mille konstrueerimisel on lähtutud maksimaalsest mahust Vahepealsed tasemed kujutavad endast erineva kiiruse ja mahuga vahemäluseadmeid (cache), mis toetuvad põhimälule, viimane omakorda kõvakettal välismälule. Tööpõhimõttelt kujutab põhimälu endast dünaamilist suvapöördusmälu (DRAM) vahemälud aga staatilisi suvapöördusmälusid (SRAM).

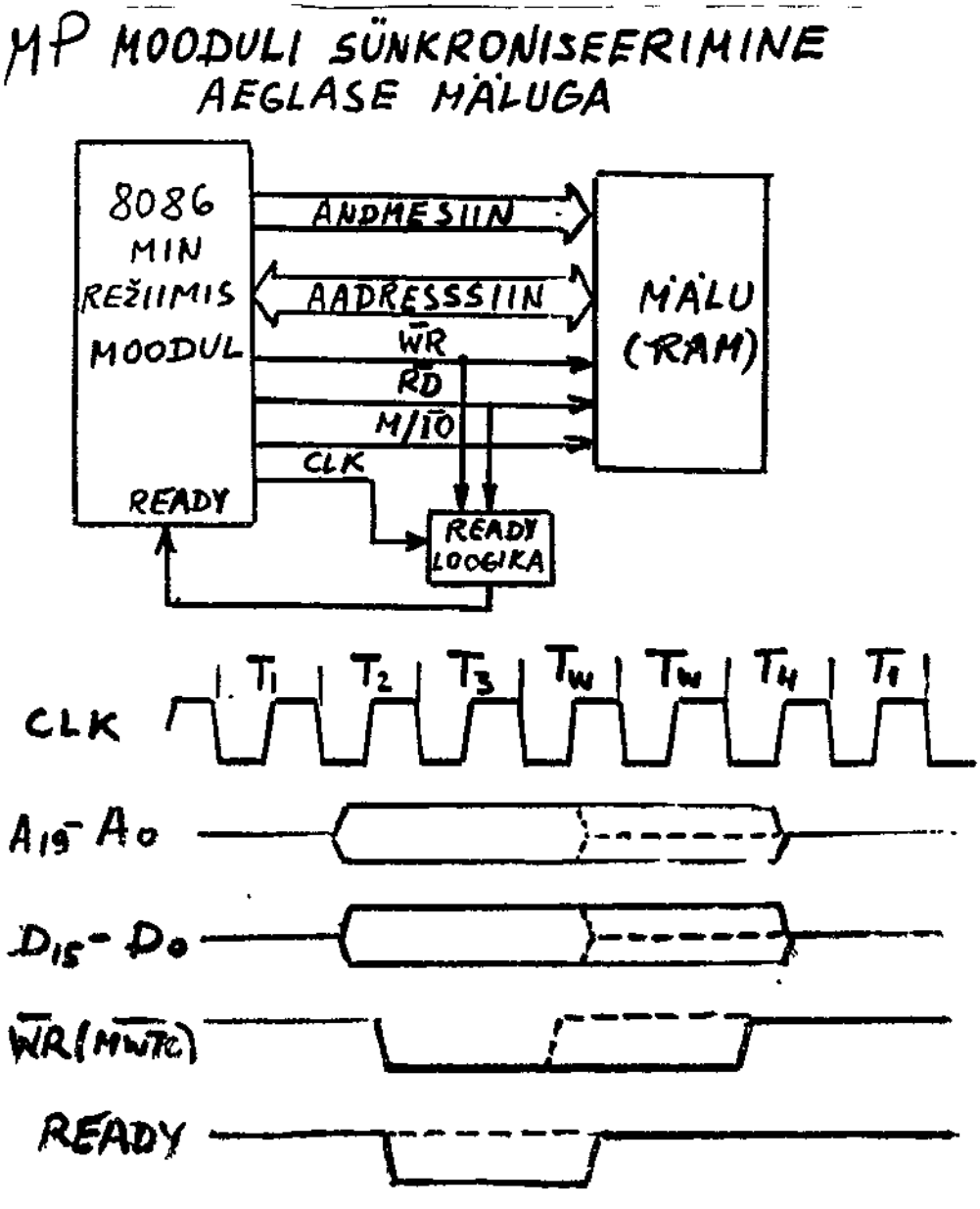
Optimeerides tasemete suhtelised andmemahud ja pöördumiskiirused on kirjeldatud mäluhierarhiat võimalik üles ehitada nii, et see moodustab protsessori jaoks piisavalt kiire pöördumisega (ilma ootetaktideta) ja praktiliselt piiramatu mahuga virtuaalse mälu. Kaasaegsetes protsessorites mahuvad protsessori kristallile juba kuni kolm vahemälutaset, mis võimaldab väga kiiret tasemetevahelist andmevahetust. Kui näiteks tsükliliselt töötav programm ja selle poolt kasutatavad andmed on juba vahemälus, ei ole protsessoril enam vajadust pöörduda suhteliselt aeglasesse põhimällu ja protsessor töötab maksimaalse jõudlusega.

Enne käivitamist aga asub programm põhimälus, salvestudes vahemälusse käivitamise faasis (esimese tsükli läbimisel), mil protsessor peab käskude ja andmete lugemiseks pöörduma suhteliselt aeglasesse põhimällu. Seejuures protsessori töö ajastamiseks aeglase mäluga, genereerib mäluseadme loogika ootetakte, kasutades selleks protsessori READY sisendit (joonis 2.2). Iga mällupöördumise algul paneb loogika READY signaali mitteaktiivseks (madal nivoo) parajasti nii mitmeks taktiperioodiks, et mälu jõuaks reageerida (antud juhul andmed salvestada). Ootetaktide lisamine protsessori ja aeglase mälu vahelise andmevahetuse sünkroniseerimiseks vähendab protsessori jõudlust ning on seetõttu ebasoovitav.

Meetodiks, mis võimaldab mälu aeglast reageerimist kompenseerida ilma protsessori jõudlust vähendamata, on aadresside mällu edastamise



Joonis 2.1

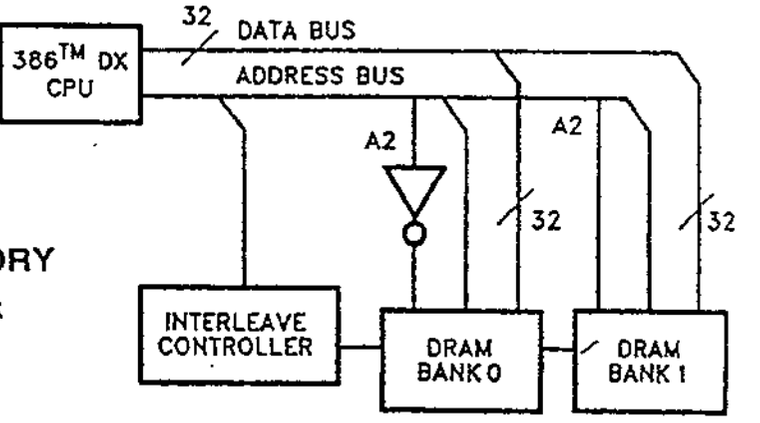


Joonis 2.2

TWO-BANK INTERLEAVED MEMORY

a) Address signal A2 selects bank

b) 32-bit datapath to each bank

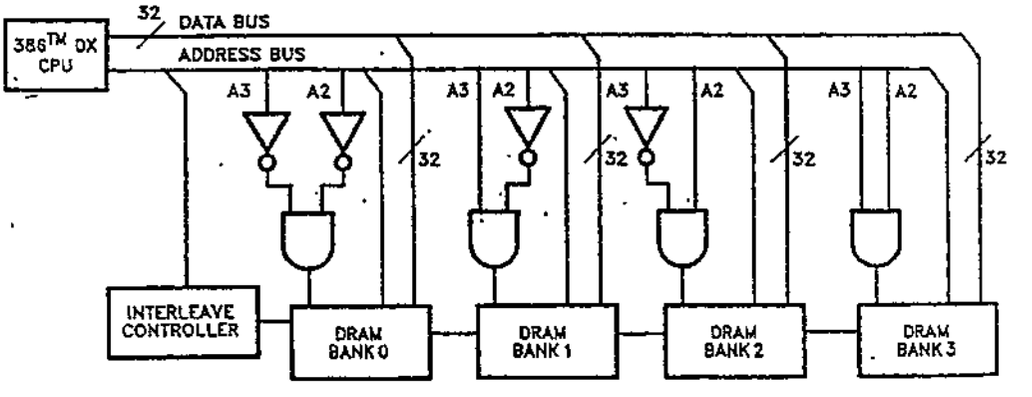


Joonis 2.3

FOUR-BANK INTERLEAVED MEMORY

a) Address signals A3 and A2 select bank

b) 32-bit datapath to each bank



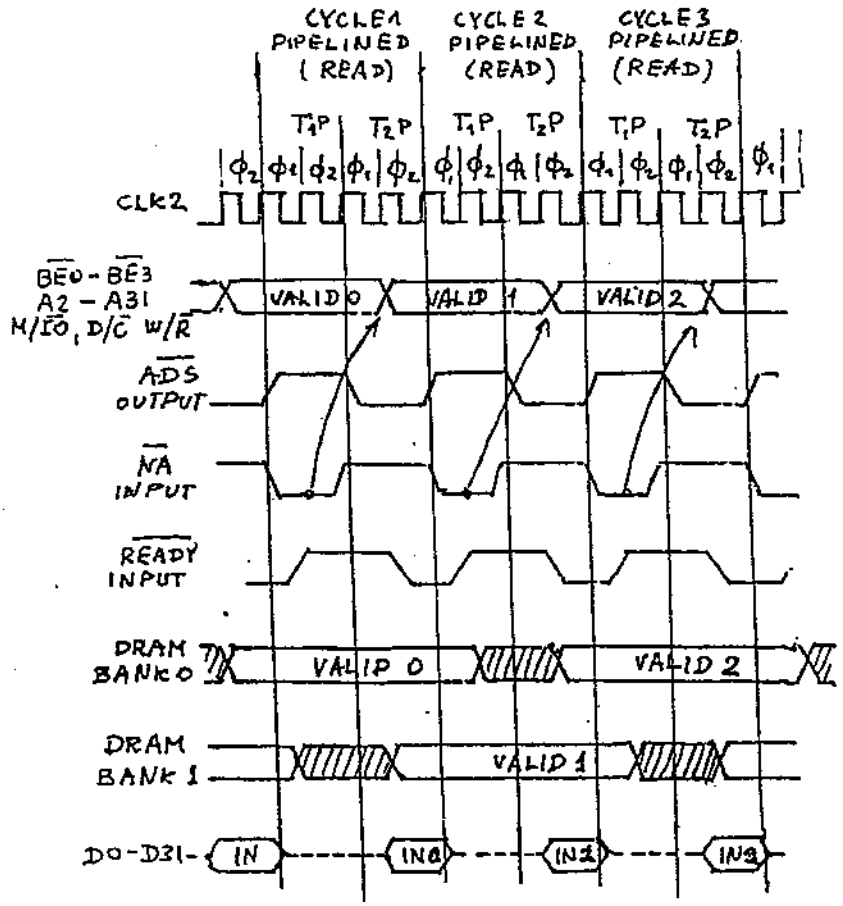
Joonis 2.6

konveiermeetod. Konveieri lihtsaim realisatsioon nõuab mälu jaotamist kaheks võrdse mahuga pangaks, millest kummagi poole pöördumine toimub vaheldumisi aadresside numeratsiooni järjekorras(joonis 2.3). Eelduseks seejuures on, et andmed (käsud või operandid) asuksid mälus vastavalt pesade numeratsioonile töötluse järjekorras. Andmed paarisaadressidel paiknevad pangas 0, paaritutel aadressidel pangas 1. Konveieri tööd illustreerivad ajadiagrammid joonisel 2.4., konveieri puudumist joonisel 2.5. Mõlemal pangal on register, milles protsessorist saabunud aadress fikseeritakse. Näiteks, vahetult pärast seda, kui pangas 0 on aadress fikseeritud, väljastab protsessor vabanenud aadressisiinile panka 1 saadetava aadressi. Samal ajal lõpetatakse lugemine pangast 0 ja andmed ilmuvad andmesiinile kust need protsessoris fikseeritakse. Pangas 1 fikseeritud aadressil toimub lugemine pangast 1 ja andmed ilmuvad andmesiinile, kust need protsessoris fikseeritakse. Samaaegselt väljastab protsessor siinile aadressi pangale 0, jne., jne. Teisiti öeldes, sel ajal, kui ühest pangast väljastatakse andmesõna siinile, toimib aadressisiinil juba teise panka suunatud aadress. Sellise pankadele toimivate aadresside osalise ajalise kattuvuse tulemuseks on see, et mälu suudab väljastada loetavaid andmeid igal pöördumisperioodil ilma ootetaktideta.

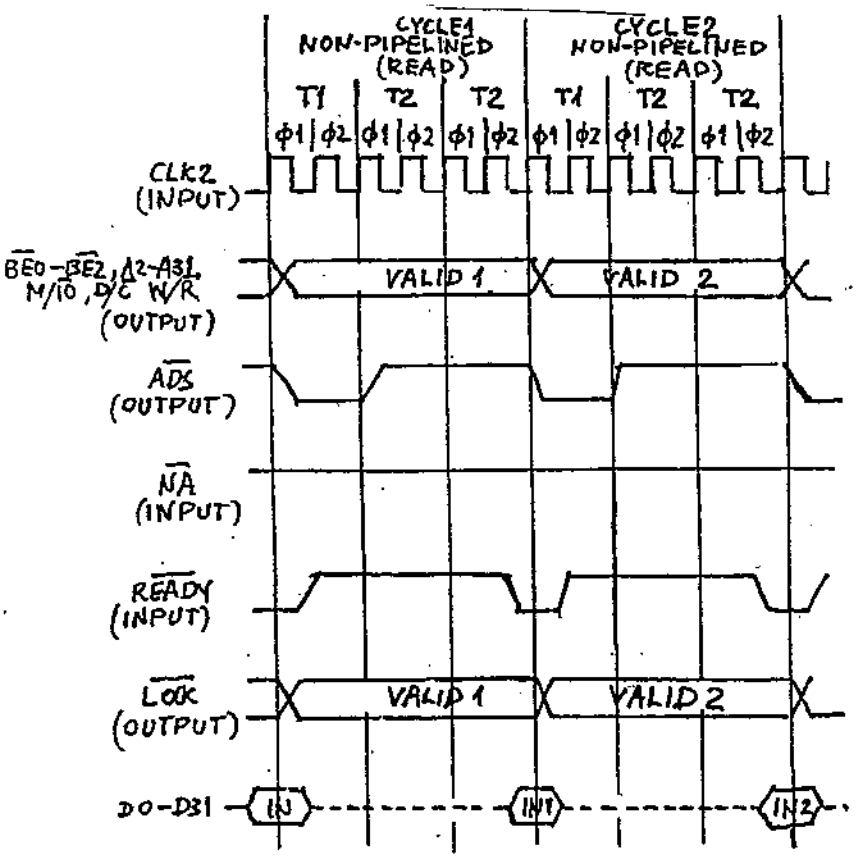
Kirjeldatud meetodit, mis põhineb vahelduval pöördumisel erinevatesse pankadesse (interleaved memory), võib laiendada näiteks neljale pangale, nagu näidatud joonisel 2.6 ning saavutada samasugust tulemust kaks korda suurema mahuga mälu puhul. Neljapangalise vahelduvpöördumisega 64 kbaidise mälu struktuur on toodud joonisel 2.7. Pankade ümberlülimine iga järgmise aadressi saabumisel toimub aadressi kahe madalama biti A0-A1 dekodeerimisel saadud signaalide abil. Dekooderi väljundid on ühendatud pankade aktiveerimise sisenditega CS.

Mälust andmete lugemise kiirendamiseks kasutatakse ka nn. pakett- e. paisklugemistsüklit (Burst Cycle Read) , mille puhul mällu saadetakse vaid andmepaketi esimese sõna aadress, välja loetakse aga peale esimese sõna veel mitu järgnevat sõna, hoides sellega aega kokku . Seda meetodit kasutatakse juhul, kui protsessor on varustatud cashe´iga , nn. cashe`I rea täitmiseks. Kui protsessor ei leia adresseeritud sõna cashe`ist, siis saadab ta aadressi edasi põhimällu, et seda sealt lugeda. Kui lugemine toimub paketttsükliga, siis koos hetkel vajaliku sõnaga loeb protsessor mälust näiteks veel kolm järgnevat sõna ja paigutab kõik neli sõna cashe`I , täites sellega cashe``I rea. Jätkates tööd , leiab protsessor järgmised sõnad cashe´ist. Pakettlugemistsükli ajadiagramme on kujutatud joonisel 2.8.

READ CYCLES WITH PIPELINED ADDRESS TIMING



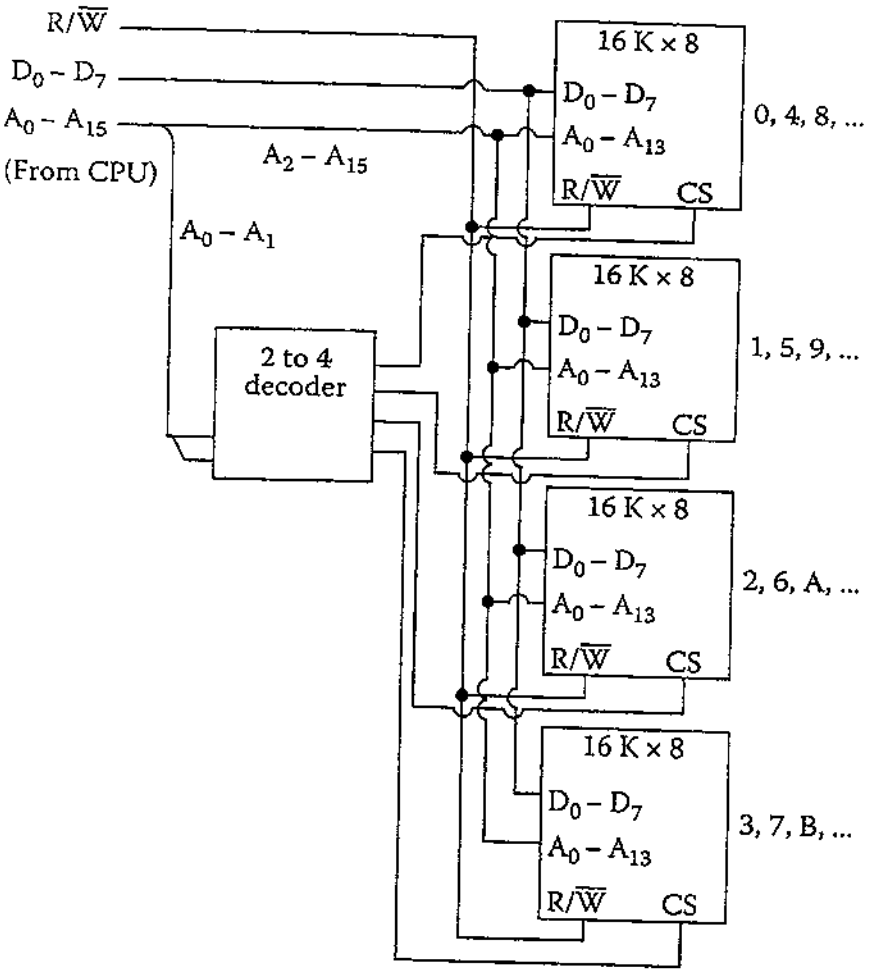
Joonis 2.4



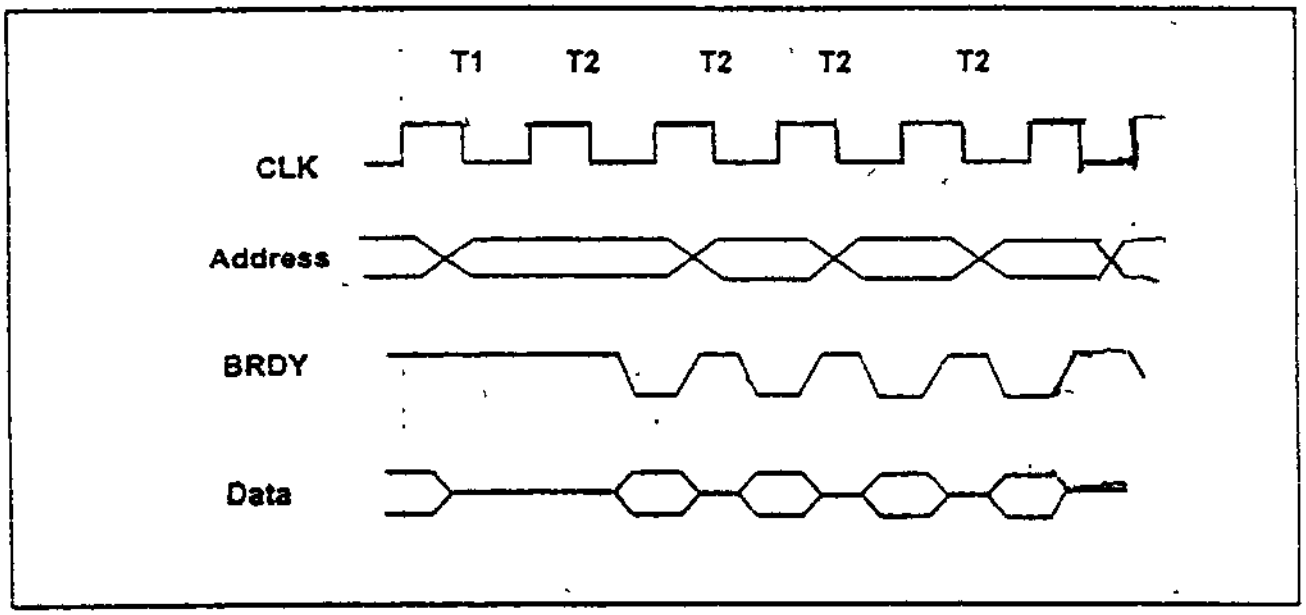
NA - NEXT ADDRESS
ADS - ADDRESS STROBE
LOCK - BUS LOCK

READ CYCLES ADDRESS TIMING WITH WAIT STATES

Joonis 2.5



Joonis 2.7



Burst Cycle Read in the 486

Joonis 2.8

3.1 Cache mälu.

Cache-i ehk tõlkes peidikmälu kontseptsioon põhineb nii töötleva informatsiooni (programmide), kui ka töödeldava informatsiooni (andmete) paiknemise lokaalsuse printsiibil. Tõepoolest programmi käsud paiknevad mälus numeratsiooni järjekorras, moodustades ühtse massiivi. Sama võib enamal juhtudel öelda ka andmete mälus paiknemise kohta. Seega on protsessori poolt järgmisena kasutatavate andmete ennustamine üsna lihtne.

Näiteks programm, töödeldes mingit andmemassiivi, täidab hetkel käsku, mis asetseb pesas *n*, siis järgmisena võtab ta käsu pesast *n+1* ning seejärel pesast *n+2* jne. Erandiks on vaid siirdekäsule järgnev käsk , juhul, kui siire toimub.

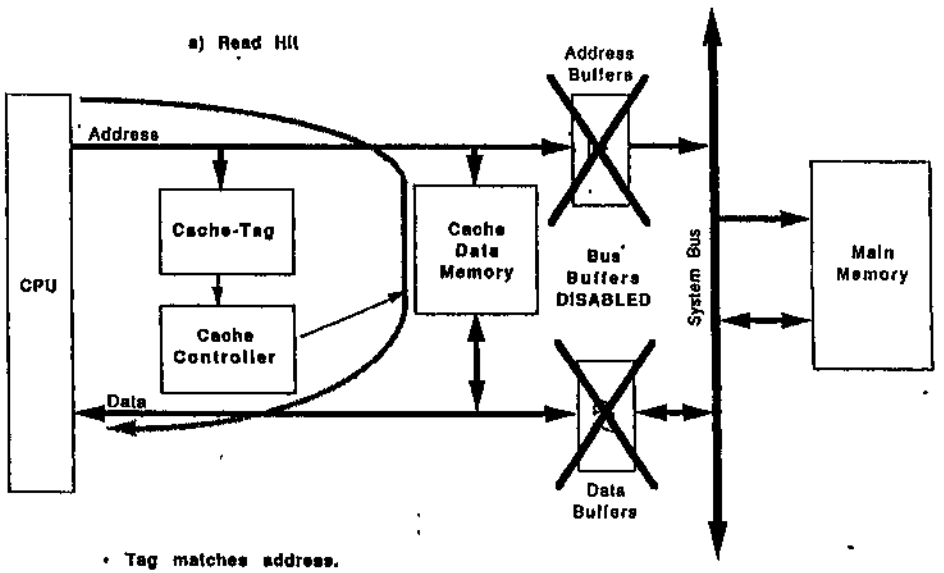
Töödeldav andmemassiiv paikneb samuti pesades, mis järgnevad üksteisele: kui mingi operand loetakse pesast *m*, siis suure tõenäosusega loetakse järgmine pesast *m+1 jne.*

Kui protsessor pöördub cache´ist ja põhimälust koosneva mälusüsteemi poole, siis esmalt cache-i kontroller võrdleb saabuvat aadressi kõigi kontrolleris olevate aadressidega, tuvastamaks, kas antud aadress on kontrolleris fikseeritud ja sel aadressil salvestatud andmed on kehtivad, ja kui on, siis on tegemist cache-i hiti ehk "tabamusega" ning toimub cache-ist lugemine (kiire lugemine). Kui aga tuvastatakse andmete puudumine cache-is (cache-i miss ehk "möödalask"), siis saadab kontroller aadressi edasi põhimällu andmete lugemiseks sealt (aeglane lugemine). Pidades silmas lokaalset paiknemist, loetakse põhimälust koos momendil vajaliku andmesõnaga ka sellel vahetult järgnev teatud sõnadeplokk, mis moodustab nn. cache-i rea (cache line) pikkusega *k* . Cache-i rida salvestatakse cache-i . Sellist lugemist nimetatakse cache-i rea täitmiseks (cache line fill).

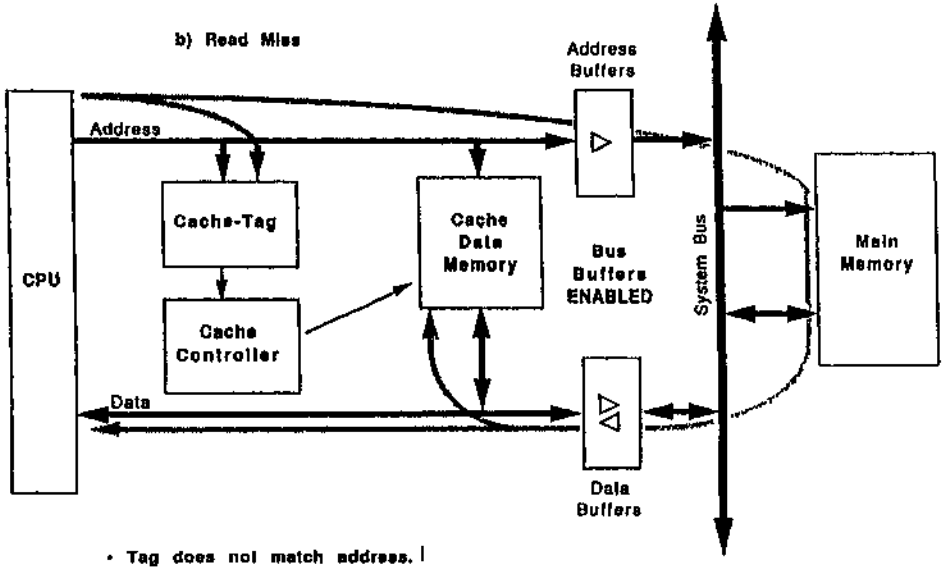
Järgmise aadressiga *n+1* mällu pöördudes leiab protsessor juba andmed cache-ist jne., kuni aadressini *n+k–1*. Edasi suunatakse pöördumine jälle põhimällu, kust loetakse cache-i rida jne. Cache-i füüsiline struktuur määrab cache-i rea pikkuse. Rea pikkus on cache-i üks olulisemaid parameetreid. Väga lühike rida eeldab sagedasi möödalaske ja põhimällu pöördumist ning seega madalat tabamuste suhet. Liig pikki ridu, aga mahutab cache-i vähe ja nende kiireks edastamiseks peab cache-i ja põhimälu vaheline andmesiin olema lai. 32-bitise protsessori puhul kasutatakse tavaliselt 2-, või 4-sõnalisi ridu, kusjuures sõna pikkuseks võetakse 16-bitti. Cache-i rea kiiremaks täitmiseks kasutatakse viimasel ajal pakettlugemist (Burst Read).

Cache-i effektiivsust protsessori jõudluse suurendamisel hinnatakse suhtarvuga:

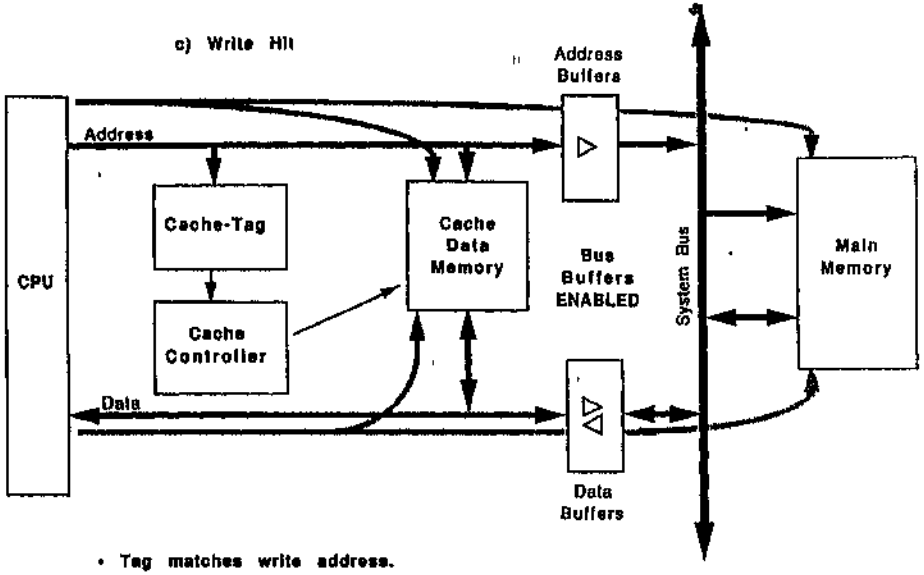
$$\frac{\text{tabamuste arv}}{\text{pöördumiste arv}} \cdot 100[\%]$$



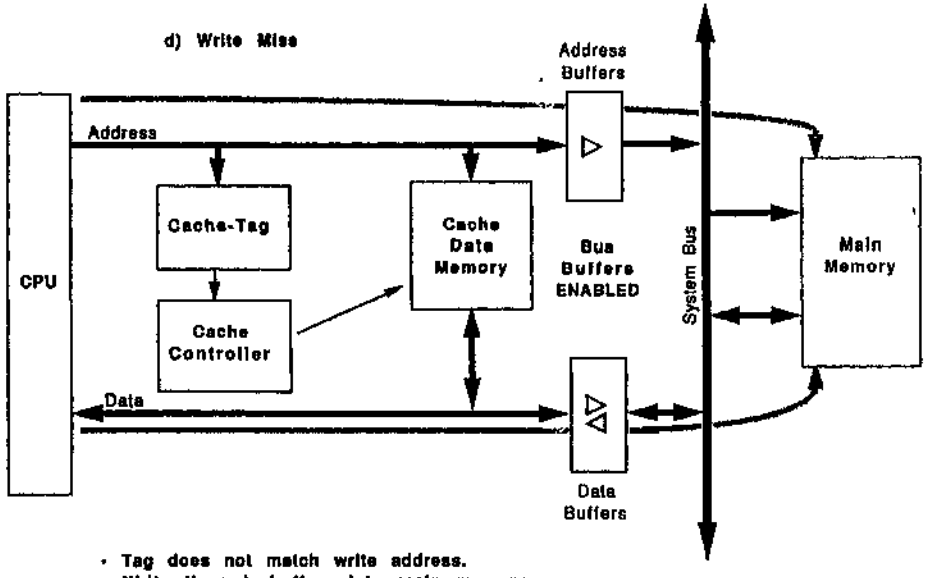
- Tag matches address.
- Read data from cache data memory to CPU.
- Disable system buffers.



- Tag does not match address.
- Read data from main memory to CPU via bus buffers.
- Write new data into cache data memory.
- Write new address into cache-tag.



- Tag matches write address.
- Write new data into cache data memory.
- Write through buffers into main memory.



- Tag does not match write address.
- Write through buffers into main memory.
- Cache data memory undisturbed.

Cache cycles: (a) Read hit. (b) Read miss. (c) Write hit. (d) Write miss.

Joonis 3.1

Seda suhtarvu mõjutavad : cache-i rea pikkus, cache-i andmemaht, füüsiline struktuur, funktsioneerimise algoritm ja täidetava programmi iseloom. Protsessor täidab programmi seda kiiremini, mida suurem on tabamuste suhtarv. Eriti kõrge cache-i kasutamise effektiivsusega töötavad iteratiivsete arvutuste tsükkelprogrammid. Selliste programmide puhul on nii programmi enda maht, kui ka töödeldavate andmete maht väike ning need mahuvad seega tervenisti cache-i. Möödalasud toimuvad sel juhul vaid tsükli esimesel läbimisel, hiljem on tabamuste arv praktiliselt 100%.
Funktsioneerimise algoritmi järgi jagunevad cache-mälud järmiselt:

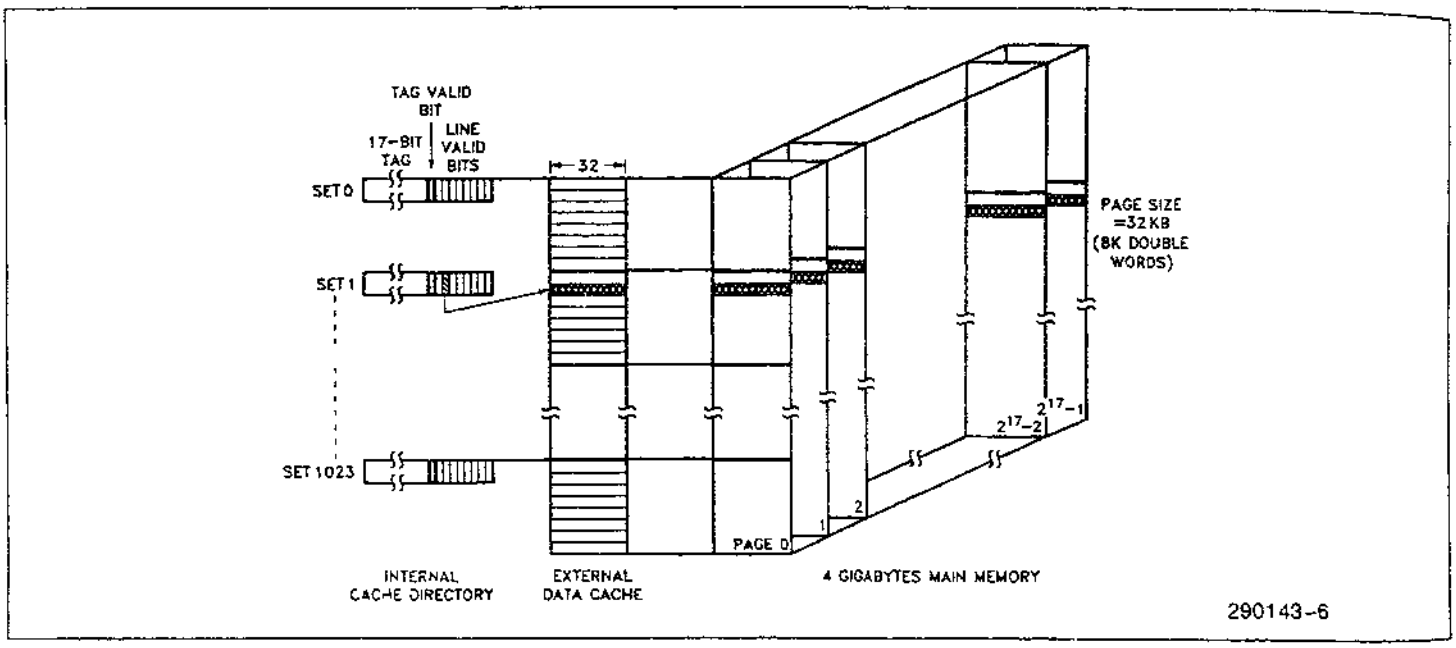
- täisassotsiatiiv (fully associative) cache;
- otsepeegeldus (direct mapped) cache;
- plokkassotsiatiiv (set assotsiative) cache;

Täisassotsiatiivcache-is võib ükskõik milline põhimäluaadress olla salvestatud cache-i aadressikataloogi mistahes pesasse. Seetõttu peavad cache-i kataloogis olema kõigi cache-i salvestatud sõnade (või ridade) täielikud aadressid. Protsessori pöördumisel mällu peab cache-i kontroller võrdlema pöördumisaadressi kõigi kataloogis olevate aadressidega, selgitamaks välja, kas andmesõna on cache-is, ja kui on , siis selle lugema. See nõuab cache-i pesade arvuga võrdse arvu võrdlustehete sooritamist ja seejuures väga kiiresti. Rahuldava kiiruse tagab aparatuurselt realiseeritud võrdlusskeem ja sedagi vaid cache-i suhteliselt mõõduka mahu puhul.

Otsepeegelduscache-i pöördumisel on vaja sooritada vaid üks võrdlustehe tänu sellele, et selles on lubatud igale põhimälus paiknevale cache-i reale üks kindel pesa cache-is. See on saavutatud aadressis indeksvälja eraldamisega cache-i ridadele. Aadressi tunnusväli (TAG) eristab antud rea kõigist teistest ridadest, mis võiksid olla salvestatud antud pessa.

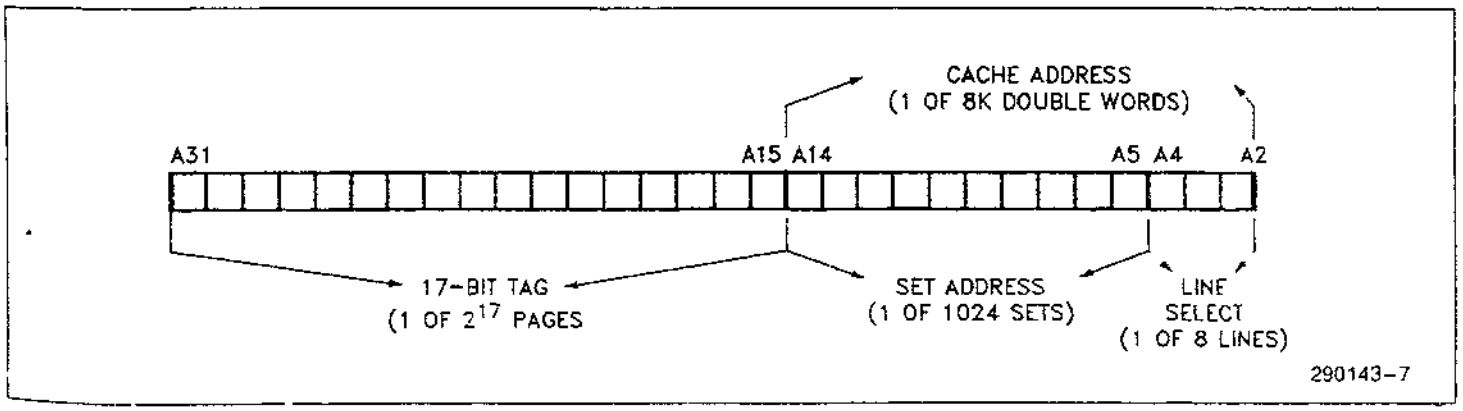
Plokkassotsiatiivcache on kas kahe-, nelja või kaheksasuunaline, mille puhul tuleb pöördumisaadressi võrrelda vastavalt kas kahe-, nelja- või kaheksa cache-is oleva aadressiga.
Nii otsepeegeldus- kui ka plokkassotsiatiivcachei üksikasjalikum tööpõhimõtte kirjeldus on toodud Intel 82385 cache-i kontrolleri näitel.
Kontrolleri otsepeegeldusreziim.
Kontroller on orienteeritud 32-tisele aadressile, mis võib adresseerida kuni 4 Gigaiti põhimälu füüsilist aadressiruumi. Cache-i mälumahuks on valitud 32 Kbaiti ehk 8K 32-bitist sõna, mis on kujutatud leheküljena (joonis 3.9.).Cache-i poolt vaadatuna jaguneb 4-Gigabaidine põhimälu 2—1analoogiliseks leheküljeks, millest iga rida peegeldub vastavale cache-i leheküljе reale. Näiteks, rida 9 mistahes leheküljelt võib

salvestada ainult 9 reale cachei leheküljel. Lehekülg on jagatud 1024-ks (0 kuni 1023) set-iks, igaühes 8 32-bitist sõna. Iga 32-bitine sõna moodustab cache-i rea. Cache-i ja põhimälu vahelise andmeedastuse ühikuks on üks rida.
Iga cache-i set omab 26-bitist sisendit kontrolleri cachei kataloogi. Need 26 bitti jagunevad järgmiselt: 17-bitine põhimälu lehekülje aadress (tag), lehekülje aadressi kehtivuse (tag valid) bitt ja kaheksa rea kehtivuse (line valid) bitti. Andmed mingis cache-i setis on kas kehtivad või kehtetud sõltuvalt aadressi kehtivuse biti olekust: 1- kehtivad, 0- kehtetud. Samaselt rea kehtivuse biti olekule 1 vastab rea kehtivus, olekule 0 rea kehtetus.
Pöördumisel Cache-i interpreteerib kontroller aadressi bitte (A2 – A31) järgmiselt (joonis 3.3): (A15 – A31) põhimälu lehekülje 17-bitine aadress, (A5 – A14) seti 10-bitine aadress ja (A2 – A4) rea 2-bitine aadress. Bitte (A2 – A4) võib vaadelda, kui cache-i 13-bitist aadressi, mis valib otse ühe 8K-st 32 –bitisest cache-i reast.
Lugemine cache-ist.
Alustades lugemist valib 10-bitine seti aadress ühe 1024-st kataloogi sisendist ja 3-bitine rea aadress valib ühe 8-st rea kehtivuse bitist. 13-bitine cache-i aadress valib vastava 32-bitise sõna cache-ist. Seejärel kontroller võrdleb saabunud 17-bitist lehekülje aadressi kataloogis oleva vastava aadressiga. Kui need ühtivad ja aadressi kehtivuse ning rea kehtivuse bitid on olekus 1, on tegemist tabamusega (hit) ja 32-bitine sõna väljastatakse andmesiinile ja sealt edasi protsessorisse.
Lugemise möödalask võib juhtuda kahel põhjusel. Esimene on rea möödalask (line miss), mis tekib siis, kui saabuv lehekülje aadress (17 bitti) ühtib kataloogis oleva vastava aadressiga, aadressi kehtivuse bitt on 1, vaid rea kehtivuse bitt on 0. Teiseks möödalasu põhjuseks võib olla kas lehekülje aadressi erinevus kataloogis olevast, või aadressi kehtivuse biti 0 olek (tag miss). Rea kehtivuse biti olek ei oma seejuures tähtsust. Mõlemal juhul suunab kontroller aadressi edasi põhimällu ja loeb sealt 32-bitise sõna (cache-i rea) ja salvestab selle chache-i. Rea möödalasu puhul kehtestatakse uus andmesõna cachei-s vastava rea kehtivuse biti asetamisega olekusse 1. Lehekülje aadressi mittevastavusest tekkinud möödalasu puhul asendatakse eelmine aadress ülesalvastamise (overwrite) teel uuega, aadressi kehtivuse bitt asetatakse 1-te, vastav rea kehtivuse bitt asetatakse samuti 1-te ja ülejäänud seitse rea kehtivuse bitti asetatakse 0-i. Üksteisele järgnevate rea möödalaskude puhul toimub ainult vastava rea kehtivuse biti sättimine. Cache-i tühjendamine (cache flush) toimub aadressi kehtivuse bittide (tag valid bits) üheaegse nullimise teel kõigis settides.
Kontrolleri plokkassotsiatiivne reziim.



Direct Mapped Cache Organization

Joonis 3.2



386 DX Address Bus Bit Fields—Direct Mapped Organization

Joonis 3.3

Joonis 3.4 kujutab kahesuunalise plokkassotsiatiivse cache-i , mis koosneb kataloogist, cache-i mälust ja 4 Gigabaidisest põhimälu ruumist. Cache-i mälu on jagatud kaheks pangaks (A ja B), kummaski 4K 32-bitist sõna. Põhimälu lehekülgede arv on kahekordistunud, lehekülje maht on kaks korda vähenenud. Iga põhimälu mingil leheküljel olev 32-bitine rida võib peegelduda nii cache-i panga A , kui panga B vastaval real. Kumbki pank jaguneb 512-ks setiks, kokku kahe panga peale 1024 setti, nagu otsepeegeldusreziimiski. Tekkinud struktuuri võib vaadelda, kui kahte poolitatud mahuga rööbitist otsepeegelduscache-i, mille kataloogid omavad kõiki juba ülal kirjeldatud atribuute. Lisandunud on vaid üks bitt, nimetusega LRU (Least Resently Used), mis tõlkes võiks tähendada "kõige varem kasutatud" ehk antud kontekstis " kõige vanemaid andmeid". Lugemise möödalasu puhul tuleb valitud setis andmeid uuendada kas A või B pangas. LRU bitt näitab kummas pangas tuleb seda teha. Cache-is olevate andmete kasutamise statistikast on teada, et kõige hiljem kasutatud (Most Resently Used) andmeid kasutatakse peagi jälle. Seepärast pannakse andmete uuendamisel pangas A või sealt lugemisel LRU bitt olekusse, mis suunaks järgmise võimaliku andmete uuendamise antud setis panka B.

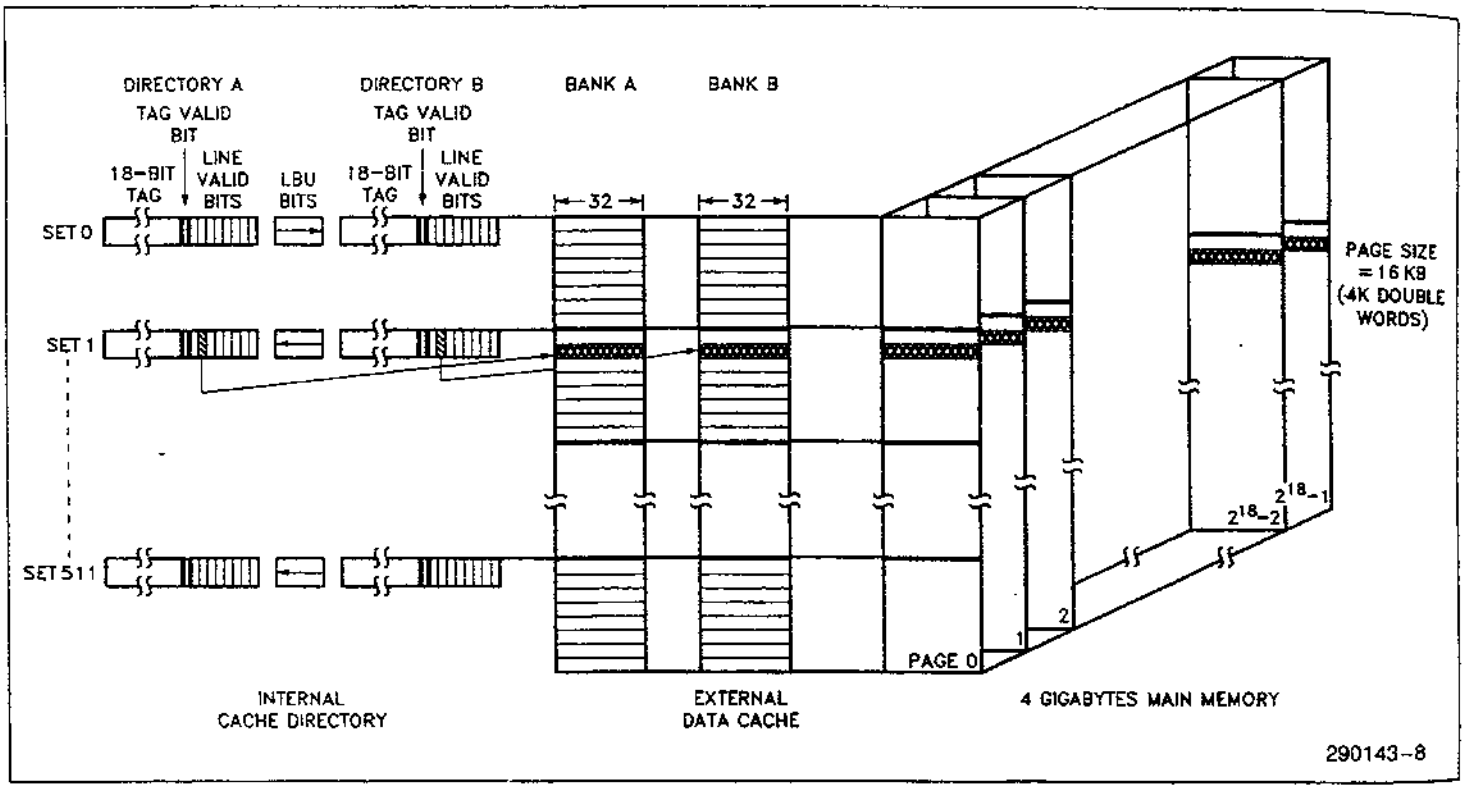
Lugemine plokkassotsiatiivsest cache-ist.

Kontroller tõlgendab 32-bitist lugemisaadressi nii, nagu näidatud joonisel 3.5. 9-bitine seti aadress (A5 – A13) valib ühe 512-st setist. Saabuvat lehekülјeaadressi (A14 – A31) võrreldakse üheaegselt kahe setis paikneva lehekülјeaadressiga, kontrollitakse mõlemat aadressi kehtivuse bitti ja mõlemat rea kehtivuse bitti. Sõltuvalt sellest, kumma panga atribuutidega võrdlus õnnestus, suunatakse sellest pangast andmesõna siinile ja sealt edasi protsessorisse. Kui andmed saadi pangast A, pannakse LRU bitt olekusse, mis viitab pangale B, kui aga tabamus oli pangas B, siis vastupidisesse olekusse, mis viitab pangale A.

Möödalask (miss) võib olla tingitud kas rea aadressi mittevastavusest pankades A ja B olevate rea kehtivuse bittide olekutele (line miss), või lehekülјeaadressi mittevastavusest valitud setis olevatele aadressidele (tag miss). Mõlemal juhul suunatakse lugemine põhimällu ja loetud cache-i rida salvestatakse panka, millele viitab LRU bitt. Uus lehekülјeaadress asendab selles pangas senini paiknenud aadressi, lehekülјeaadressi kehtivuse bitt seatakse olekusse 1, rea kehtivuse bitt pannakse samuti olekusse 1 ja ülejäänud seitse bitti nullitakse. LRU biti olek muudetakse vastupidiseks, et see viitaks teise panka, sest värskeim cache-i rida paikneb nüüd antud pangas.

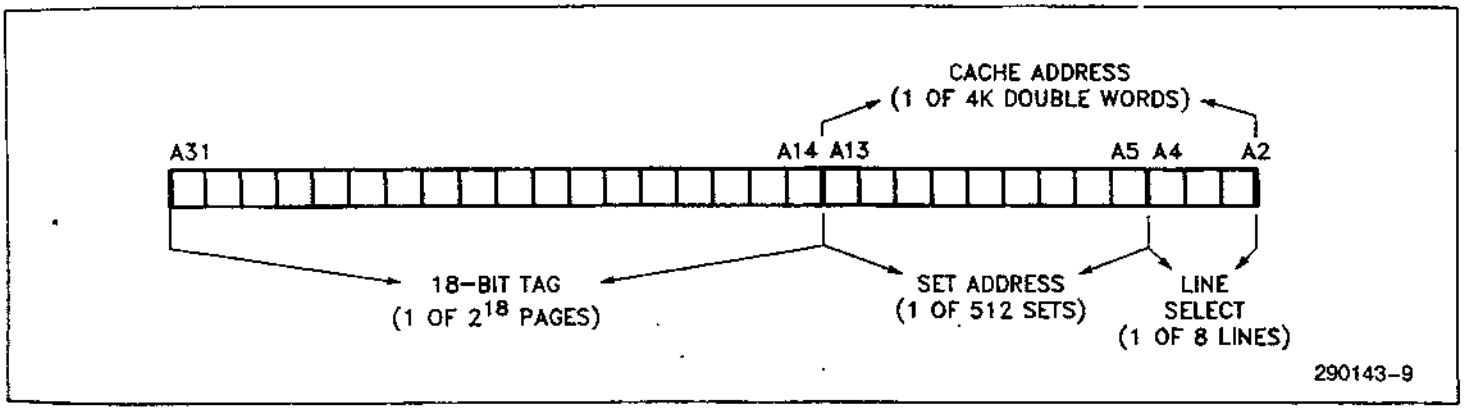
Salvestamine cache-i.

Protsessorites (näiteks Pentiumis) kasutatakse esimese taseme cache-ina kahte eraldi cache-i: käskude (e. programmi) jaoks ning andmete jaoks.



Two-Way Set Associative Cache Organization

Joonis 3.4



386 DX Address Bus Bit Fields—Two-Way Set Associative Organization

Joonis 3.5

Kuna protsessor programmi täites käske ei salvesta, vaid ainult loeb, siis programmicache on konstrueeritud ainult lugemiseks ja on seega lihtsama ehitusega. Andmecache seevastu peab lisaks lugemisele omama ka salvestamise võimalust, mis teeb ka selle konstrueerimise keerukamaks. Cache-i salvestamisel on kasutusel kaks meetodit: läbivsalvestus (write through) ja tagasisalvestus (write back).

Läbivsalvestus.

Üldse toimub cache-i salvestamine ainult tabamuse puhul, s.o. juhul kui pöördumisaadress eksisteerib cache-is ja on kehtiv. Samaaegselt salvestatakse sõna nii cache-i, kui ka samal aadressil põhimällu. Seejuures salvestusprotsess mõlemasse mällu algab üheaegselt, kuid määravaks osutub salvestamise kestus põhimällu, kui aeglasemasse seadmesse. Salvestuse tulemusena on nii cache-i pesas , kui ka põhimälu sama aadressiga pesas uus andmesõna, s.o. vastavate pesade sisu on koherentne. Kuid aega kulub selleks sama palju kui tavaliselt põhimällu salvestamisel ja seega cache-i kasutamise kiirendav toime puudub.

Läbivsalvestuse eeliseks on küll kontrolleri aparatuurse realisatsiooni lihtsus, kuid peamiseks puuduseks on see, et cache-i salvestuse kiirus jääb põhimällu salvestuse kiiruse tasemele. Selle puuduse kõrvaldamiseks tuleb cache-i kontroller varustada puhvriga, kus salvestatav sõna ja selle aadress kiiresti fikseeritakse ja salvestusprotsessi põhimällu sooritab cache-i kontroller ning protsessor vabaneb edasisest põhimällu salvestamisest, jätkates tööd järgmise käsuga.

Tagasisalvestus.

See meetod nõuab keerukamat aparatuurset realisatsiooni, kuid see eest on effektiivsem juhul, kui programmis on sageli salvestusi. Tabamuse puhul salvestatakse sõna ainult chache-i pesasse, põhimälu vastava pesa sisu jääb seejuures uuendamata. Seega toimub salvestus kiiresti. Vastavasse põhimälu pesasse jääb vananenud sõna seniks, kui cache-i rida, kuhu salvestati uus sõna, ei hakata asendama uuega põhimälust lugemise teel. Et seejuures salvestamisel muudetud sõna kaduma ei läheks, tuleb see cache-ist põhimällu salvestada enne, kui cache-is rida uuega asendatakse. Et märgistada cache-i rida kuhu on salvestatud, peab kataloogi tunnuse registris iga rea kohta olema täiendav bit, mis antud ritta salvestamisel läheb olekusse 1. Biti selle oleku järgi salvestatakse muudetud sõna sellest reast põhimällu enne selle uuega asendamist.

Salvestuse möödalasu puhul toimub salvestus ainult põhimällu.

Kokkuvõtteks võib öelda, et cacheis hoitakse jooksvalt hetke kõige intensiivsemalt kasutatava informatsiooni (käskude ja andmete) võimalikult identset koopiat põhimälus samadel aadressidel paiknevast informatsioonist. Selle koopia identsus ongi cachei koherentsus. Cacheist lugemise möödalaskudel tõmbab kontroller põhimälust cachei ridade koopiaid paigutades neid cachei. Seejuures koherentsus säilib. Läbiv

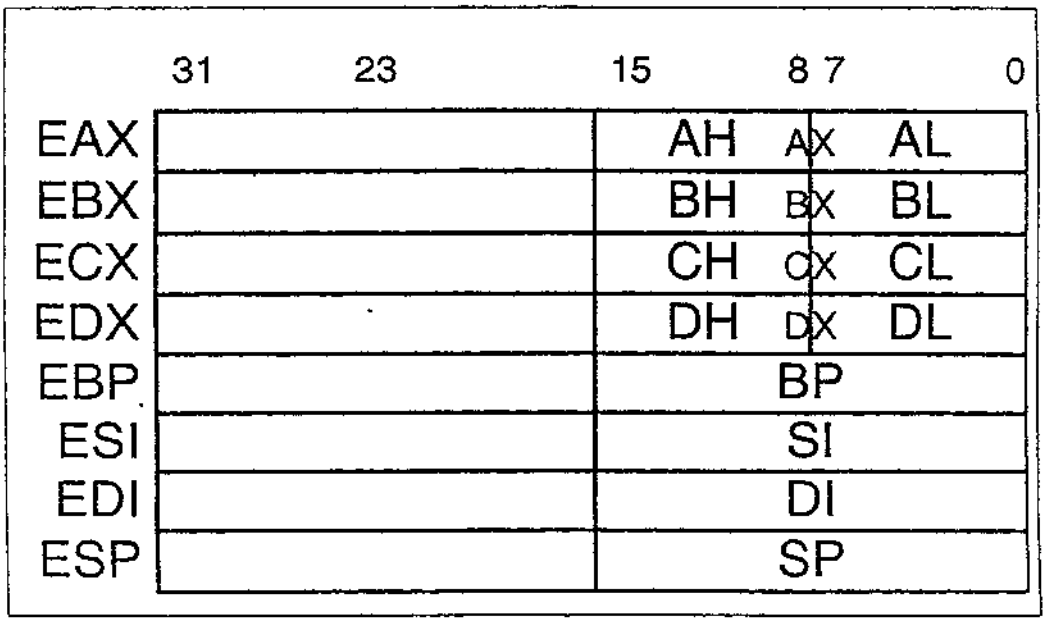
salvestus säilitab samuti koherentsuse. Tagasisalvestus rikub koherentsust, kuid ei sea selle puhul ainult cache-is olevaid andmeid kaotsimineku ohtu, kuna muudetud rea cache-ist kõrvaldamise eel salvestatakse muudetud sõna põhimällu. Oht tekib siis, kui protsessor jagab ühist põhimälu mõne teise aktiivse seadmega, näiteks kõige sagedamini otsemällupöörduskontrolleriga (DMA – Direct Memory Access), mis juhib andmevahetust kiirete välisseadmetega. Kui otsemällupöördustsükkel satub lugema pesast, mille cache-is olev koopia on vahepeal uuendatud, satub näiteks välismällu (kõvakettale) vananenud andmesõna. Et seda ei juhtuks, jälgib cache-i kontroller DMA lugemistsüklite aadresse. Kui DMA aadress ühtib pesa aadressiga, mille sisu on cache-is uuendatud (seda sündmust signaliseerib cache-i salvestamise bitt), jätkatakse DMA lugemistsüklit alles pärast seda, kui pesa uus sisu on cache-ist põhimällu kopeeritud.
DMA kontroller salvestab põhimällu ka uusi andmeid. Selle tulemusena tekivad põhimälus andmed, mis erinevad cache-i vastavates pesades olevatest andmetest. Antud juhul osutuvad vananenuks cache-is olevad andmed. Et seda ei juhtuks jälgib cache-i kontroller DMA salvestustsüklite aadresse erilise jälgimissiini (snoop bus) kaudu. Kui DMA tsükli aadress ühtib cache-is oleva sõna aadressiga , nullitakse kataloogis selle rea kehtivuse bitt, milles sõna paikneb. Sellega tõkestatakse protsessoril vananenud andmete lugemine cacheist ja protsessor peab pöörduma põhimällu, kus asub värske andmesõna.

4.1 CISC protsessorid

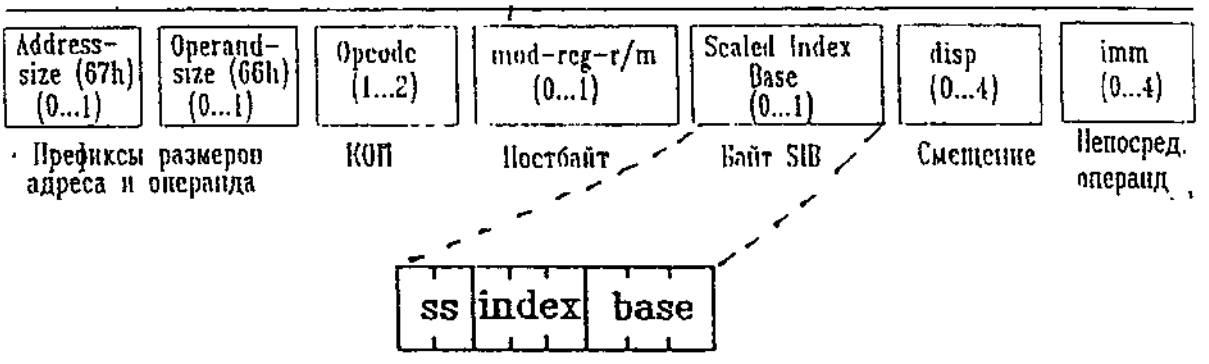
CISC –tüüpi protsessorite musternäiteks on firma Intel toodang alates mudelist 8086, mis valmis 1978.a. ja lõpetades Pentium 4-ga. Kuna Inteli protsessorid on olnud ja on ka praegu personaalarvutites enim kasutatavad, siis on loogiline käsitleda CISC protsessorite arenguga seotud probleeme just nende näitel.
Intel Itanium ja uusimad kahe-, nelja- ja enam tuumalised protsessorid on juba midagi enamat, kui klassikalised CISC protsessorid.
Intel 8086 ja 80286 olid 16-bitised, mis tähendab, et nende täisarvulise aritmeetika seade töötles paralleelkoodis 16-bitiseid (ka 8-bitiseid) operande , registrite pikkus ja välise andmesiini laius oli 16 bitti. Aadressisiini laius oli 8086-l 20 bitti, mis võimaldas adresseerida kuni 2 = 1 Mbait mälu, 80286-l 24 bitti ja seega võimaldas adresseerida kuni 2 = 16 Mbaiti mälu. 16-bitist arhitektuuri hakati hiljem tähistama lühendiga IA-16 (Intel Arcitecture).
Intel 80386-s (aastal 1985) võeti kasutusele 32-bitine arhitektuur IA-32, milles üldregistrid, ALU ja väline andmesiin on 32-bitised, kusjuures säilis nii 8-bitine, kui ka 16- bitine aritmeetika. IA-32 kehtib kõigis järgnevates mudelites kuni Pentium 4-ni. Seega moodustavad Inteli erineva põlvkonna protsessorid arhitektuurse "pärilikkusega" omavahel seotud "dünastia", mis paistab välja üldregistrite ühildatud kujutisest joonisel 4.1. Kuigi alates Pentiumist laiendati välist andmesiini 64 bitini, siis tehti seda andmevahetuse tõhustamiseks põhimäluga. Arhitektuurilt jäid protsessorid ikkagi 32-bitisteks. 64-bitine andmesiin võimaldab ühe pöördumisega mällu kas lugeda või salvestada kahte 32 –bitist, nelja 16-bitist või kaheksat 8-bitist operandi, või lugeda käskusid kuni 64-bitise pikkusega.
Arhitektuur hõlmab registreid, käsustikku, mis koosneb suurest hulgast erineva vorminguga käskudest, erinevaid andmeformaate ja nende adresseerimisviise, ühesõnaga kõike seda, mida peab teadma programmeerija. Teiste sõnadega arhitektuur kujutab endast arvuti mudelit programmeerija jaoks. IA-32 üldistatud käsuformaat on toodud joonisel 4.2.
Kuigi IA-32 protsessorid on arhitektuurilt sarnased ja seega on nad ühtselt käsitletavad, siis oma mikroarhitektuurilt võivad nad olla üsnagi erinevad. Mikroarhitektuur kirjeldab protsessori aparatuuri (riistvara) tööd operatsioonskeemide, loogikaskeemide ja mikroprogrammide tasemel, olles seega protsessori riistvara konstruktori mudeliks. Alljärgnevalt vaatlemegi tagasivaates erineva põlvkonna Inteli protsessorite mikroarhitektuuri iseärasusi ja peamisi arenguprobleeme alates Pentiumist.

4.2 Pentium

1993.a. valminud Pentiumi mikroarhitektuuri kodeeritud tähiseks on P5, mille kõige olulisemaks iseärasuseks on kahest rööbitiseks tööks mõeldud konveierist (U- ja V-konveier) koosnev täisarvuliste operandide töötlemise seade (joonis 4.3). U-konveieri jätkuna töötab P5 koosseisus konveierireziimis ka ujukomaprotsessor, milles on nii liitmis-lahutamis, kui

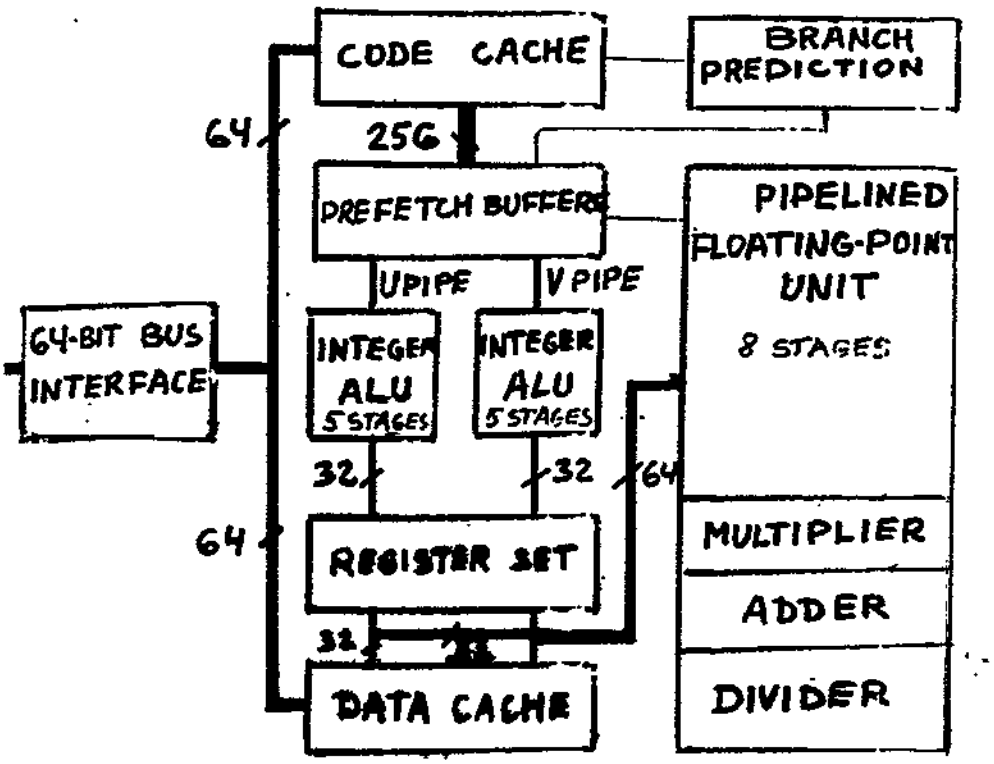


Joonis 4.1



Joonis 4.2

PENTIUM 1993

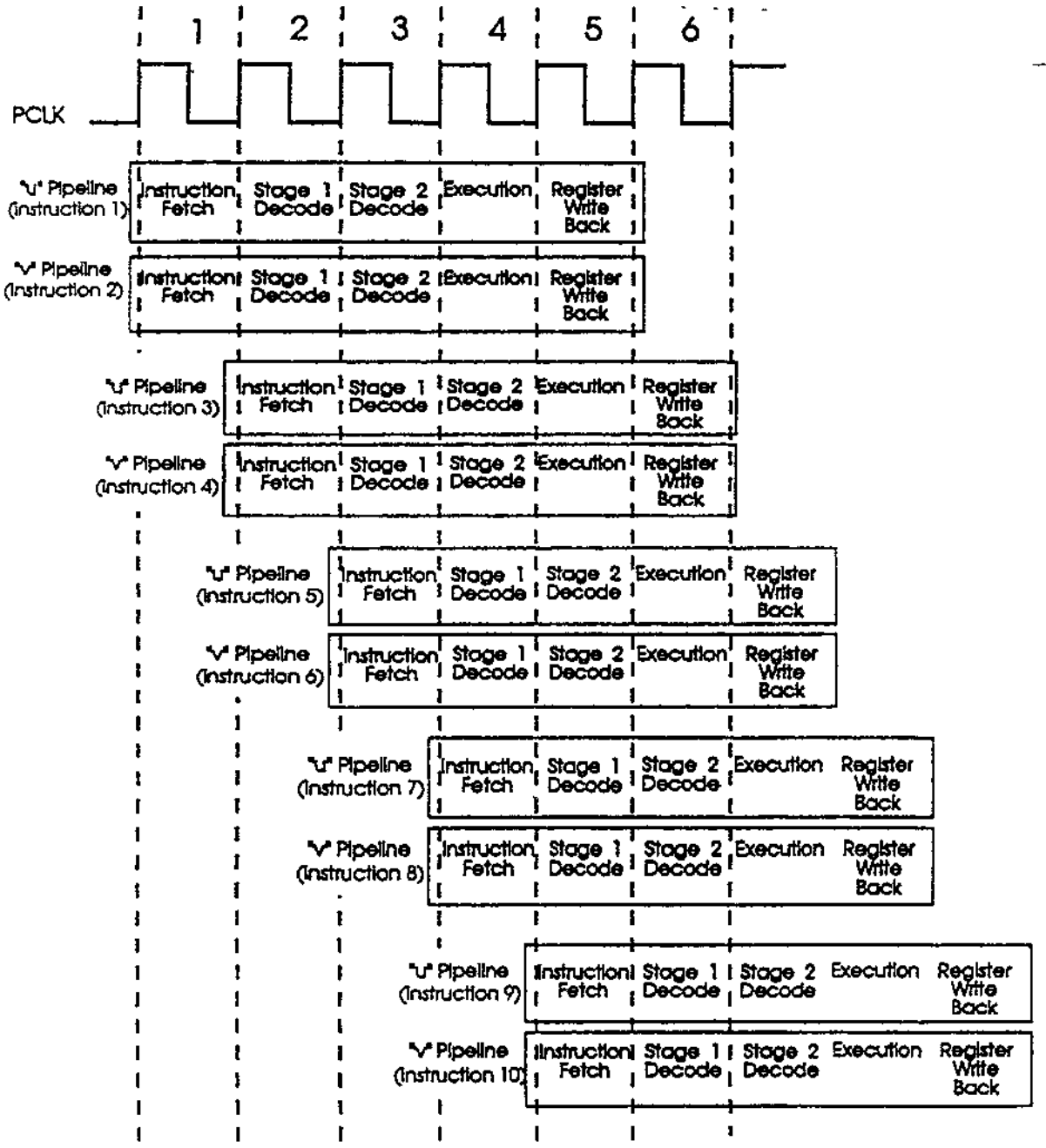


U ja V konveier	PF	D1	D2	E	WB			
Ujuvkoma konveier	PF	D1	D2	OF	X1	X2	WF	ER

PF (Prefetch) – käsu lugemine cache'ist
D1 (First decode) – dekodeerimise 1. aste
D2 (Second decode) – dekodeerimise 2. aste
E (Execute) – käsu täitmine
WB (Write back) – tulemuse salvestamine (registrisse)
OF (Operand fetch) – ujuvkoma operandide lugemine cache'ist või/ja registritest.
X1 (First execute) – käsu täitmise 1. aste
X2 (second execute) – käsu täitmise 2. aste.
WF (Write float) – tulemuse salvestamine ujuvkoma registritesse.
ER (Error reporting) – vearaport: vea avastamisel käivitab veatöötluse. Määrab ujuvkoma olek-registri sisu.

Joonis 4.3

kakorrutamise ja jagamise seadmed realiseeritud aparatuurselt selleks, et suurendada jõudlust. Oluliseks täienduseks, võrreldes eelmise mudeliga (i480) on ka eraldi vahemälud (cashe'id) käskudele ja andmetele, mis võimaldab üheaegselt käskusid ja andmeid lugeda. Käskude cashe on mõeldud ainult lugemiseks ja seetõttu ehituselt lihtsam, kuna protsessor programmi täitmise käigus loeb ja käske kunagi neid ei salvesta. P5 sisaldab ka programmi hargnemise dünaamilise ennustamise plokki. Kuna Pentium omab rohkem kui ühte töötlusseadet, siis kuulub ta ka nn. superskalaarsete protsessorite klassi.
Kahe konveieri ühte protsessorisse paigutamise eesmärgiks oli tõsta selle jõudlust ideaaljuhul kahekordseks. Kui ühe konveieriga protsessor suutis täita parimal juhul igal taktil ühe käsu, siis kaks konveierit rööbiti töötades peaksid suutma täita igal taktil kaks käsku. Pentiumi viieastmeliste konveierite rööbititööd illustreerib joonis 4.4. Nii ideaalselt töötaksid konveierid vaid siis, kui kõik konveieritesse laaditud käskude paarid oleksid ühesuguse formaadiga ja võrdse täitmise ajaga kõigis konveieri astmetes, mida Pentiumi keeruka käsustiku tõttu on kui mitte võimatu siis väga raske saavutada. Kui ühe konveieri mingis astmes kulub käsu sammu täitmiseks rohkem aega, kui teise konveieri samas astmes, siis teise konveieri aste peab esimese järel ootama.
Teiseks konveierite efektiivset kasutamist segavaks põhjuseks on väike protsessori registrite arv, mis ei võimalda kompaileril koostada programmi alati nii , et konveieritesse laaditud käskude paaril ei tekiks täitmisel konflikti sel pinnal, et mõlemad kasutavad üht ja sama registrit. Kui selline konflikt tekib, siis üks konveieritest peatub, rööptöö katkeb ja käsud täidetakse üksteise järel.
Kolmandaks põhjuseks on täidetavate käsupaaride omavaheline andmesõltuvus: ühe käsu operandiks on teise käsu täitmise tulemus. Sel juhul tuleb käsud samuti täita üksteise järel ja üks konveieritest seisab kui teine töötab. Kompaileri töö programmi koostamisel võtab kirjeldatud ohtusid arvesse ja püüab neid leevendada, kuid sõltuvust registritest on nii väikse arvu registrite puhul (ainult 8 registrit) väga raske vältida.
Ujukoma aritmeetika käskude täitmisest võtab osa ainult U-konveier koos kolme täiendava lisaastmega. Samal ajal V-konveier seisab.
Ülalöeldu põhjal saab teha järelduse, et Pentiumi kahe konveieriga superskalaarne mikroarhitektuur on kauge esialgselt kavandatud ideaalist peamiselt registrite puuduliku arvu ja keeruka käsustiku tõttu.



The Pentium Processor's Dual Instruction Pipeline

Joonis 4.4

5.1 P6 mikroarhitektuur.

Protsessori Pentium Pro jaoks, mis oli Pentiumi järglane, töötati välja uus mikroarhitektuur koodnimetusega P6, mis on kujutatud joonisel 5.1. Lühidalt kirjeldatuna täidab Pentium Pro käskusid järgmiselt:

1. Võtab mälust käske programmiga määratud järjekorras.
2. Dekodeerib neid programmiga määratud järjekorras ja teisendab üheks või enamaks fikseeritud pikkusega RISC käsuks ehk mikrooperatsiooniks ehk mikro-opsiks
3. Paigutab mikro-ops-id käskude tabelisse programmiga määratud järjekorras.
4. Selle punktini toimus käskude töötlemise protsess esialgses, programmiga määratud järjekorras, mille tulemusena käsud asendati mikro-opsidega. Protsessor hakkab täitma mikro-opse sellises järjekorras, millises täitmiseks vajalikud operandid ja tehete sooritusplokid ühele või teisele mikro-opsile kättesaadavaks osutuvad.
5. Niipea, kui tehete sooritamise tulemused selguvad, paigutab protsessor need protsessori registritesse programmiga määratud järjekorras.

Käsuvõtu kirjeldamiseks on joonisel 5.2 kujutatud 64-baidine mälupiirkond (pesad 0h kuni 3Fh), milles paikneb rida käske. Käskude pikkused on erinevad: ühebaidised, kahebaidised, kolmebaidised jne. Oletame, et käsuvõtu blokk laadib programmicache-ist 32-baidise rea (pesad 0h kuni 1Fh) käsuvõtu puhvrisse (joonis 5. 3). Oletame, et toodud programmilõigu esimesele käsule viitab hargnemisennustuse ploki poolt väljastatud hargnemiskäsu JUMP sihtaadress. Sihtaadressile eelnevad baidid selles reas tühistatakse, kui mittevajalikud.

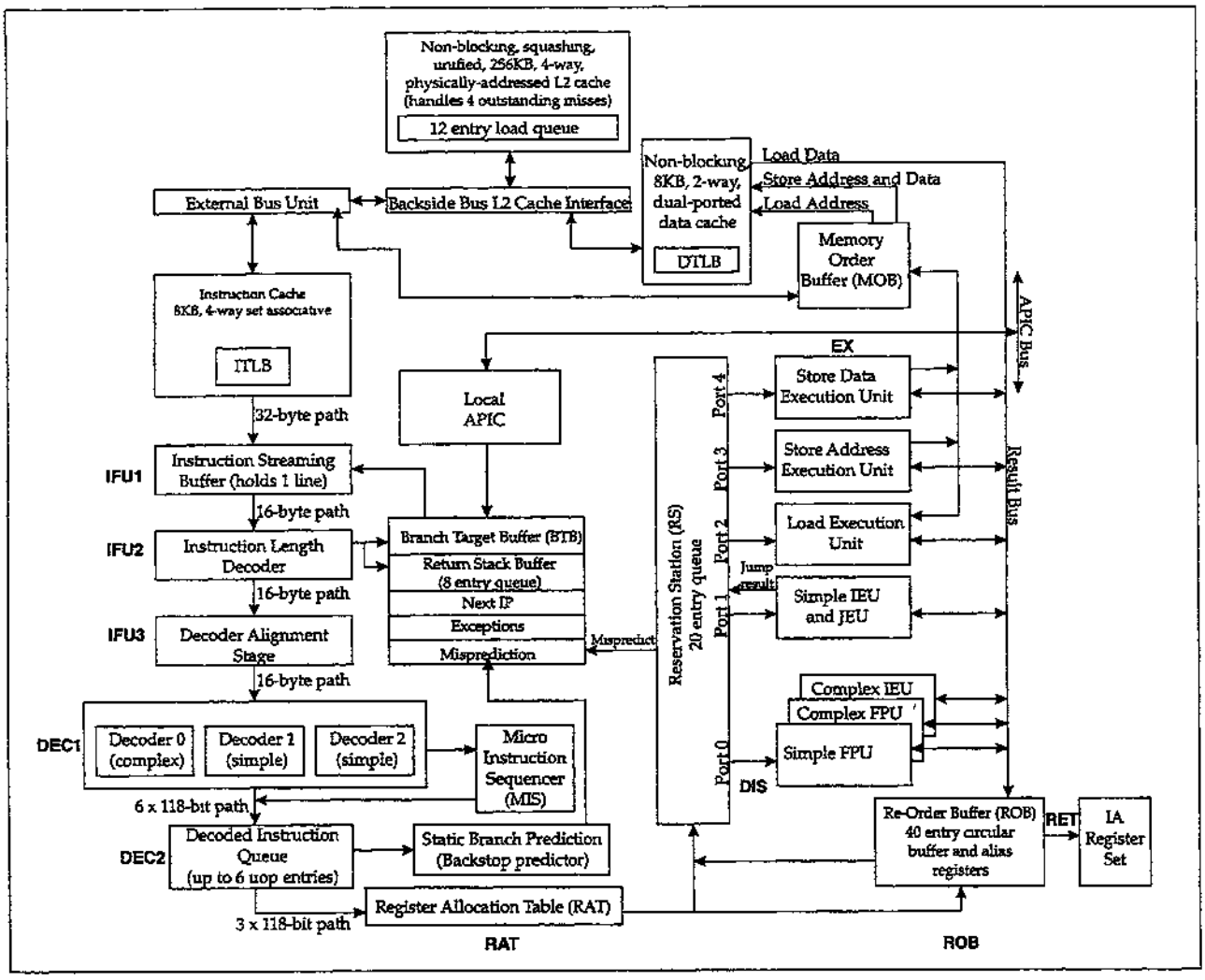
Käskude töötlemine P6 mikroarhitektuuri 11-astmelisel konveieril (joonis5.4) toimub astmete kaupa nii nagu alljärgnevalt kirjeldatud.

Kolm esimest astet (IFU1, IFU2 ja IFU3) sooritavad käsuvõtu järgmiselt:

IUF1: Käsuvõtu puhvris fikseeritud 32-baidine cache-i rida suunatakse 16 baidi kaupa edasi järgmisse astmesse ja seejärel täidetakse puhver järgmise 32-baidise reaga.

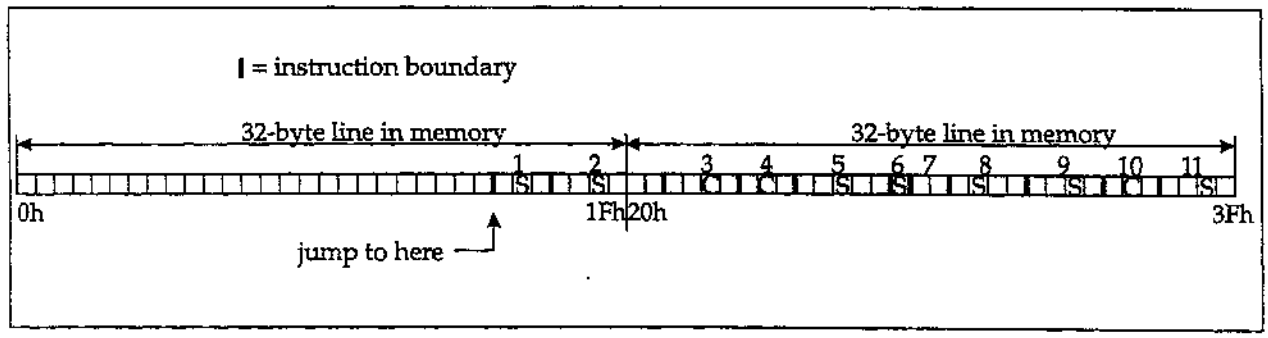
IUF2: Identifitseeritakse ja märgistatakse käskude piirid esimeses 16-baidises plokis. Kontrollitakse, kas plokis on hargnemiskäskusid. Kui on, siis saadetakse nende käskude aadressid hargnemisennustuse ploki puhvrisse (Branche Target Buffer) hargnemise ennustamiseks.

IFU3: Protsessor reastab kolm järjekordset IA käsku vastavalt nende keerukusele kolme dekooderiga (joonis 5.5), nende teisendamiseks ühepikkusteks käskudeks e. mikro-opsideks, pikkusega 118 bitti. Iga mikro-ops jaguneb neljaks väljaks, milleš paiknevad: käsukood, kaks lähteaadressi, sihtaadress.



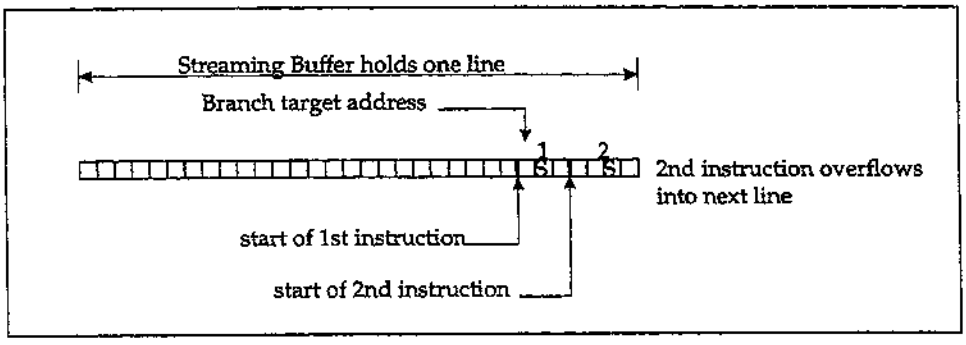
Joonis 5 .1

Example Instructions in Memory



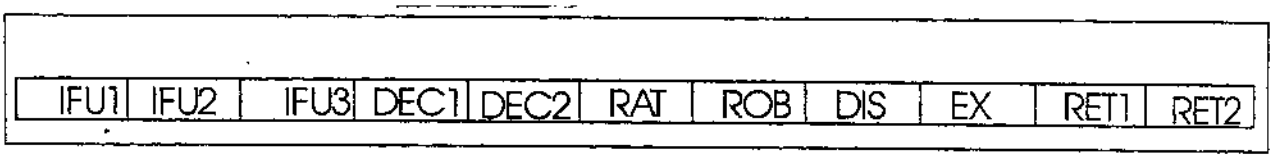
Joonis 5.2

Contents of Prefetch Streaming Buffer Immediately after Line Fetched



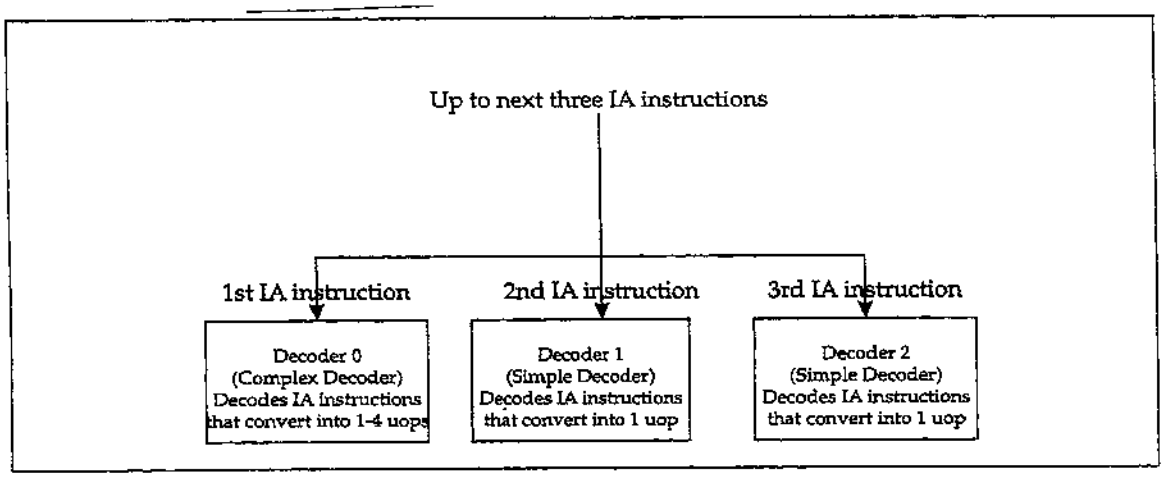
Joonis 5.3

Instruction Pipeline



Joonis 5.4

The Three Instruction Decoders



Joonis 5.5

Esimene kolmest dekooderist (dekooder 0) on keerukas (complex), mis võib teisendada IA käsu, sõltuvalt selle keerukusest üheks kuni neljaks mikro-opsiks. Teine (dekooder 1) ja kolmas (dekooder 2) dekooder teisendavad ainult lihtsaid IA käske üheks mikro-opsiks.
Suurem osa IA käske on teisendatavad ühe mikro-opsilisteks. Nendeks on:

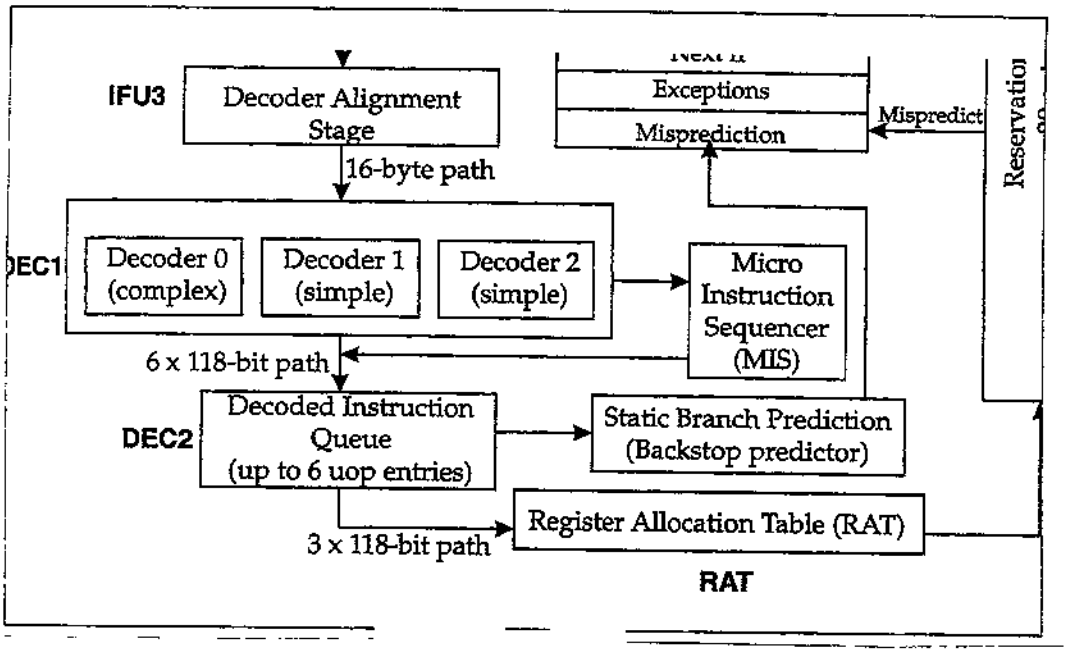
- register – register tüüpi käsud;
- mälust lugemise käsud;

Kaheks mikro-opsiks teisendatakse :

- mällu salvestamise käsud;
- mälust lugemise/ modifitseerimise käsud;

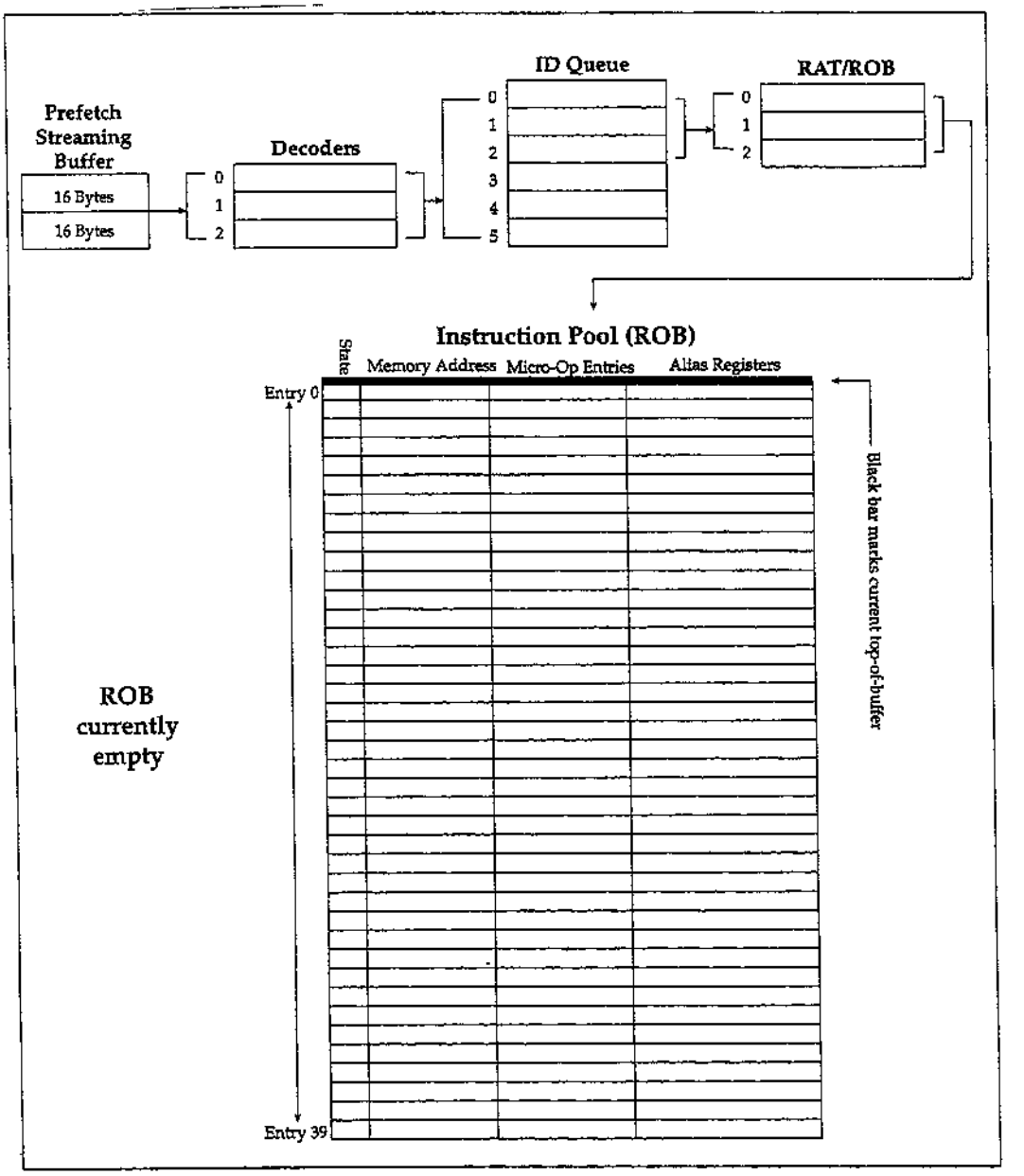
Kaheks või kolmeks mikro-opsiks teisendatakse register/ modifitseerimise/ mällu salvestamise käsud.
Neljaks mikro-opsiks teisendatakse mälust lugemise/ modifitseerimise /mällu salvestamise käsud;
Kasutades IFU2-s paika pandud käskude piirimärgendeid, reastatakse käsud IFU3-s dekooderitega vastavalt nende keerukusele nihutamise teel ja edastatakse astmesse DEC1.
DEC1: Teisendab IA käsud mikro-opsideks järgnevalt: Keerukas dekooder (dekooder 0) suudab dekodeerida mistahes IA käsu, mille pikkus ei ületa 7 baiti, üheks kuni neljaks mikro-opsiks. Lihtsad dekooderid (dekooder 1, dekooder 2) teisendavad mistahes IA käsu , pikksega mitte üle 7 baidi, üheks mikro-opsiks. Mõned IA käsud tuleb teisendada rohkemaks, kui neljaks mikro-opsiks. Need käsud suunatakse teisendamiseks mikroprogrammide plokki, mis põhineb mikrokäskude püsimälul (ROM), kus paiknevad käskudele vastavad mikro-opside jadad igaühes viis ja enam mikro-opsi.
DEC2: Edastab mikro-opsid dekodeeritud käskude järjekorda (ID Queue) (joonis 5. 6), üheaegselt kuni 6 mikro-opsi , igaüks 118 bitti, esialgses IA käskude järjekorras. Kui mõni mikro-opsidest vastab teisendatud hargnemiskäsule, kasutatakse staatilist hargnemisennustust, tegemaks kindlaks kas mikro-opsi täitmisel hargnemine toimub või mitte.
RAT: (Register Allocation Table) Kuna IA registrite komplektis on ainult 16 adresseeritavat reristrit: EAX, EBX, ECX, EDX, ESI, EDI, EBP ja ESP ning FP0....FP7, siis üksteisele järgnevate käskude üheaegne täitmine on sageli võimatu. Et seda vältida, on protsessoris 40 peidetud registrit, milliseid kasutatakse IA registrite asendamiseks tehete sooritamise ajal. See võimaldab käskude vahelisi registrisõltuvusi vältida ja seega käske üheaegselt täita. RAT astmes valib protsessor, milliseid 40-st asendusregistrist kasutada ümberjärjestuspuhvris (ReOrder Buffer – ROB). Mikro-opsid kasutavad tehete sooritamisel registreid mis asuvad ROB-is.
ROB: (ReOrder Buffer) Ümberjärjestuspuhver (joonis 5.7) Pärast dekooderite poolt mikro-opside väljastamist ja nende aadresside fikseerimist RAT-i astmes paigutatakse mikro-opsid ROB-i programmiga määratud järjekorras kolme kaupa igal taktil.

Decoders and the Micro Instruction Sequencer (MIS)



Joonis 5.6

The ROB, the RS and the Execution Units



Joonis 5.7

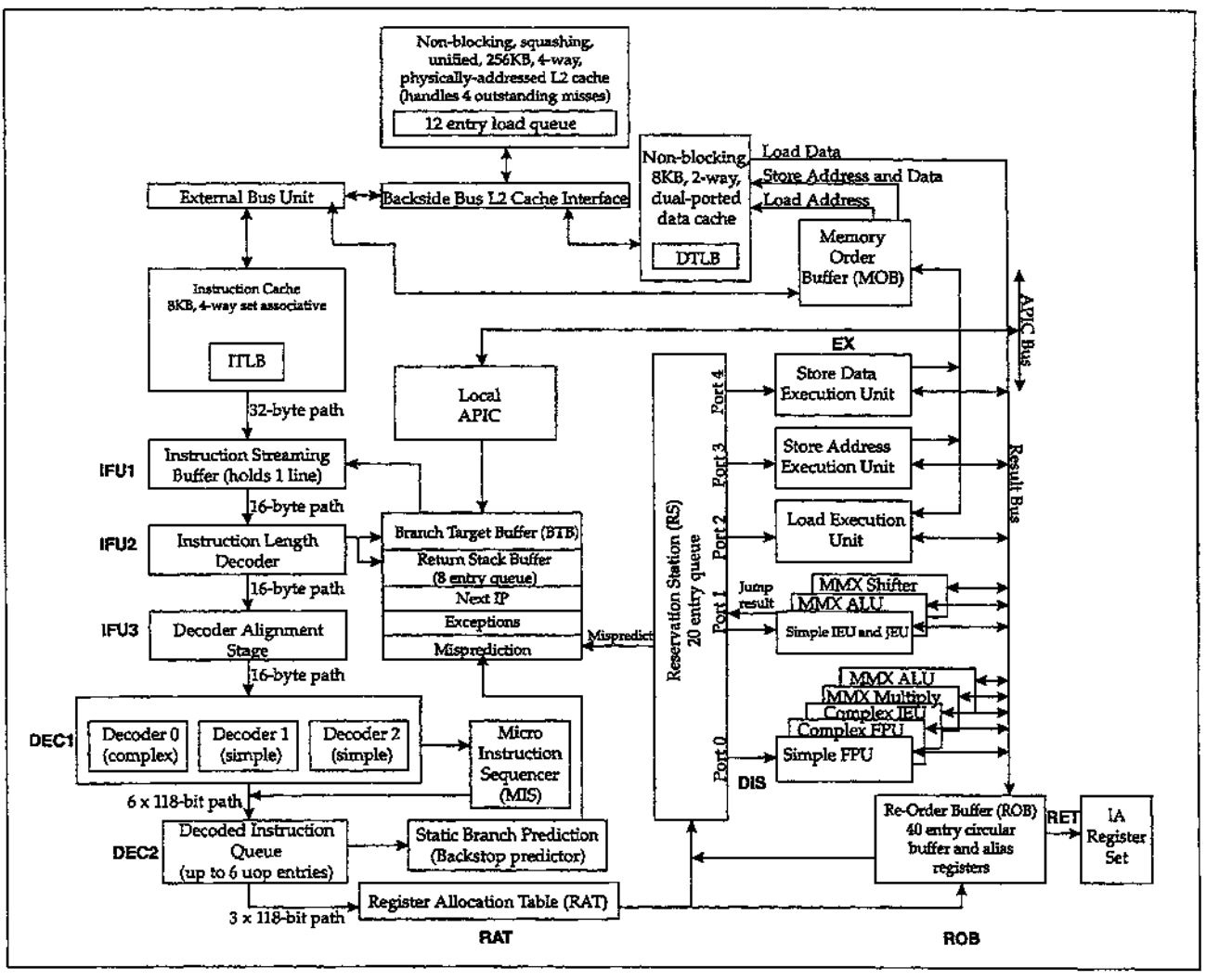
DISP ja **EX:** Varustusjaam (Reservation Station – RS) kopeerib üheaegselt kuni viis mikro-opsi ja paneb need järjekorda kuni nende sisestamiseni vastavatesse sooritusüksustesse. Mikro-opside varustusjaamast sooritusüksustesse väljastamise kriteeriumiks on täitmiseks vajalike lähteandmete ja vastavate sooritus üksuste kättesaadavus. Seega ei pea mikro-opse täitma mingis kindlas järjekorras. Mingi mikro- opsi sooritamise tulemus salvestatakse ROBi samasse positsiooni, kus paikneb mikro-ops.
RET1: Toimub täidetud mikro-opside märgistamine nende väljastamiseks konveierilt. Selle astme loogika kontrollib ROB-is pidevalt kolme kõige varem täidetud mikro-opsi olekut, et tabada hetke, millal kõik kolm on väljastamiseks märgistatud.
RET2: Toimub mikro-opside konveierist taandamine kolme kaupa korraga programmiga määratud järjekorras. Mikro-opside täitmise tulemused kopeeritakse ROB-ist protsessori IA registritesse ja vastavad mikro-opsid kustutatakse.

5.2 PENTIUM II

Protsessor Pentium II on üles ehitatud P6 mikroarhitektuuril. Seepärast joonisel 5.8 kujutatud struktuurskeem on äravahetamiseni sarnane Pentium Pro vastava skeemiga. Ainsaks erinevuseks on Pentium II-le lisatud täiendavad plokid maatriksarvutuste sooritamiseks täisarvuliste operandide gruppidega, põhimõttel- üks käsk , mitu operandi (SIMD- Single Instruction Multiple Data). Operandide grupiviisiliseks töötlemiseks on protsessori käsustikule lisatud üle viiekümne täiendava käsu. Maatriksarvutuste grupiviisiline sooritamine võimaldab oluliselt kiirendada multimeedia informatsiooni (piltide ja helide) töötlemist. Seepärast nimetatakse Pentium II –s kasutusele võetud aparatuurseid ja programseid vahendeid multimeedia laiendiks (Multimedia Extension – MMX).
Töödeldavate operandide formaadid võivad olla 8- , 16- , 32- ja 64- bitised, nagu näidatud joonisel 5.9. Multimeedia käskude täitmisel kasutatakse ujukoma aritmeetikaploki 80-bitiseid registreid FPU0....FP7 64 biti ulatuses, tähistades need MMX0.....MMX7 (joonis 5.10). Vastavalt operandide formaadile jagatakse registrid alamregistriteks.

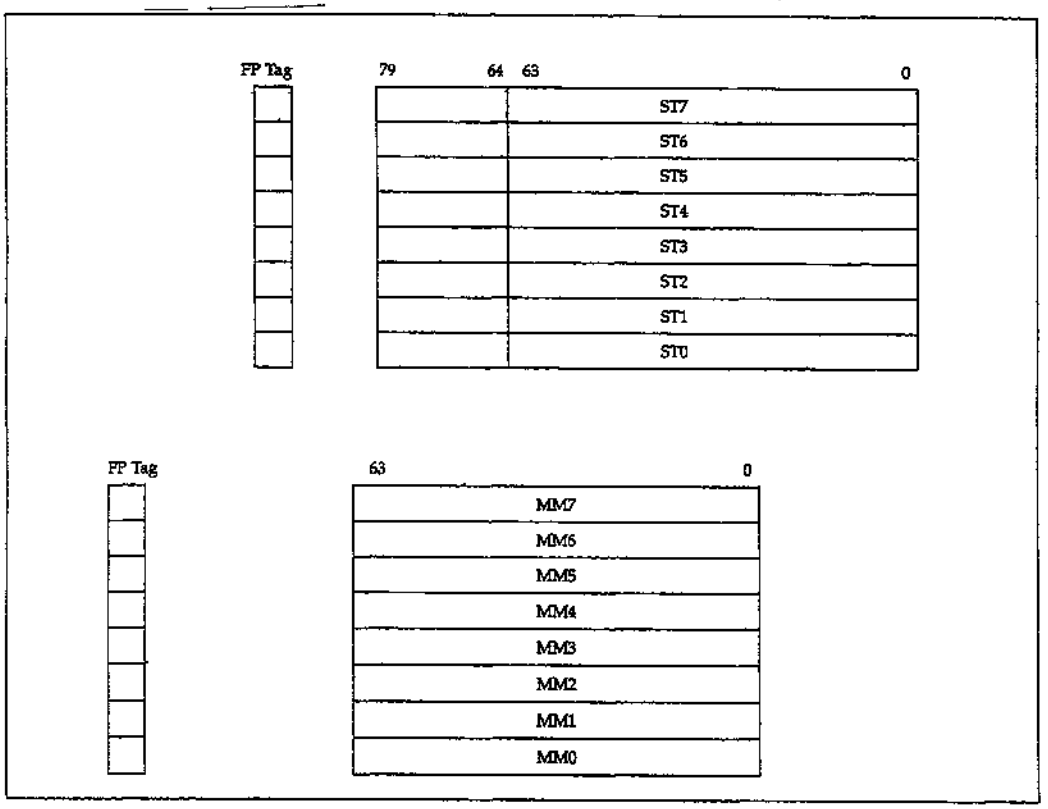
5.3 PENTIUM III

Protsessor Pentium III ilmus turule 1999. a. alguses ja kätkeb endas kõiki P6 mikroarhitektuuri omadusi parendatud ja laiendatud kujul. Märkimisväärseks täienduseks Pentium III-s võrreldes eelmiste protsessoritega on ujukoma formaadis multimeedia andmete töötlusplokk koos 70 uue käsuga, mis võimaldab töödelda video, audio ja 3D graafika andmeid.
Operandide formaadiks on valitud 32-bitine IEEE-754 standardi järgne ühekordse täpsusega ujukoma arvude formaat, mille mantiss on 23-bitine ja astmenäitaja 8-bitine ning märgi bitt.

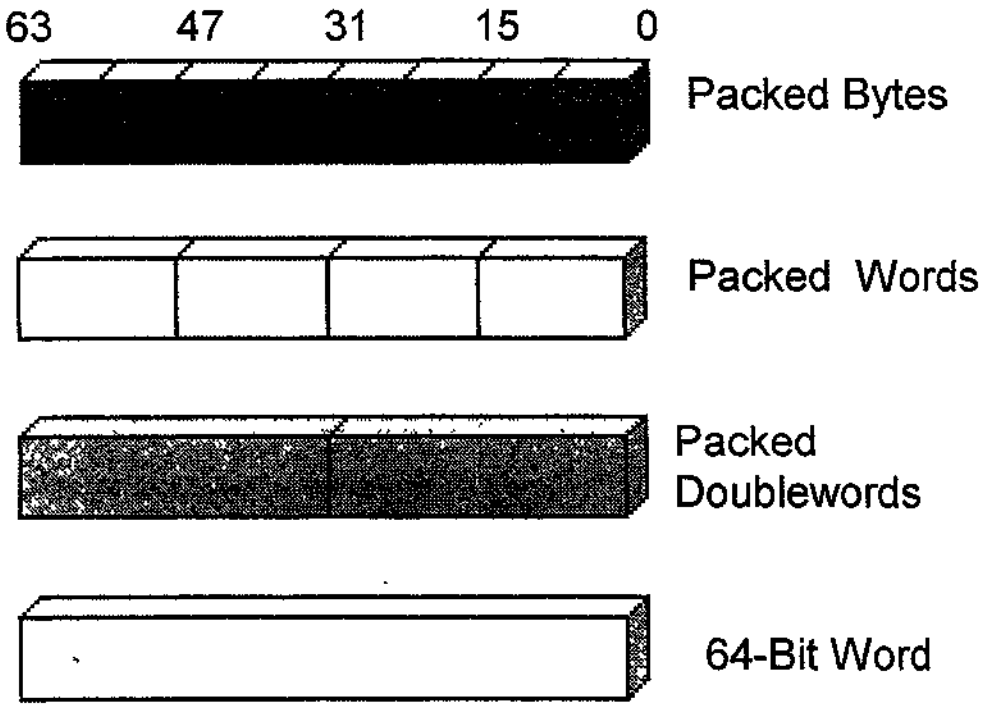

Non-blocking, squashing, unified, 256KB, 4-way, physically-addressed L2 cache (handles 4 outstanding misses)
12 entry load queue
Non-blocking, 8KB, 2-way, dual-ported data cache
DTLB
Load Data
Store Address and Data
Load Address
Memory Order Buffer (MOB)
External Bus Unit
Backside Bus L2 Cache Interface
Instruction Cache 8KB, 4-way set associative
ITLB
32-byte path
Local APIC
APIC Bus
EX
Store Data Execution Unit
Store Address Execution Unit
Load Execution Unit
MMX Shifter
MMX ALU
Simple IEU and JEU
MMX ALU
MMX Multiply
Complex IEU
Complex FPU
Simple FPU
Port 4
Port 3
Port 2
Port 1
Port 0
Jump result
Result Bus
Reservation Station (RS) 20 entry queue
IFU1
Instruction Streaming Buffer (holds 1 line)
16-byte path
IFU2
Instruction Length Decoder
16-byte path
IFU3
Decoder Alignment Stage
16-byte path
Branch Target Buffer (BTB)
Return Stack Buffer (8 entry queue)
Next IP
Exceptions
Misprediction
Mispredict
DEC1
Decoder 0 (complex)
Decoder 1 (simple)
Decoder 2 (simple)
Micro Instruction Sequencer (MIS)
6 x 118-bit path
DEC2
Decoded Instruction Queue (up to 6 uop entries)
Static Branch Prediction (Backstop predictor)
3 x 118-bit path
Register Allocation Table (RAT)
RAT
DIS
Re-Order Buffer (ROB) 40 entry circular buffer and alias registers
RET
IA Register Set
ROB

Joonis 5.8

MMX Registers are Mapped Over FP Registers

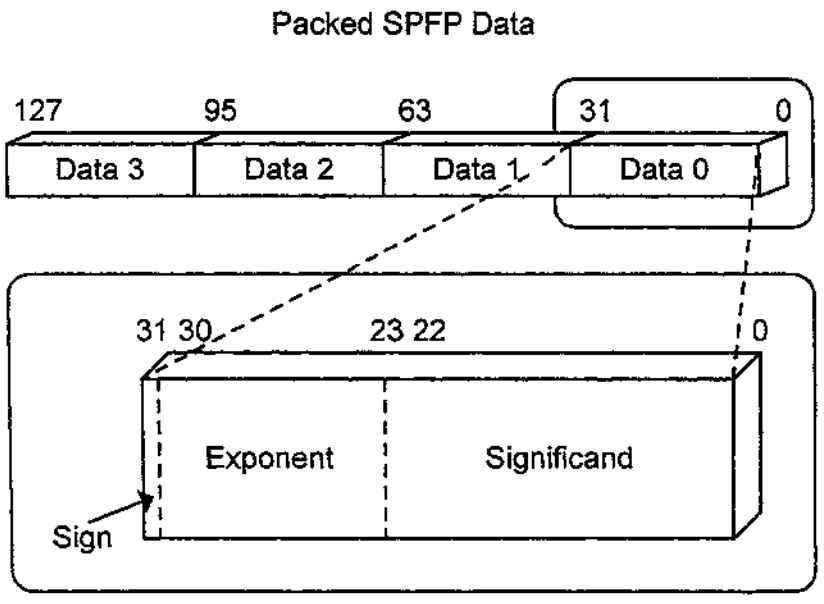


Joonis 5.10



Joonis 5.9

Üks SIMD ujukoma käsk võib sooritada tehte üheaegselt nelja 32-bitise ühekordse täpsusega ujukoma arvuga (SPFP- Single- Precision Floating-Point number), mis moodustab 128-bitise andmeühiku (joonis 11).
SIMD käskude täitmiseks SPFP andmetega on protsessoris kaheksa uut 128-bitist XMM registrit, mis on tähistatud kui XMM0.....XMM7.



SPFP data representation format.

Joonis 5.11

5.4 Pentium 4

Protsessor Pentium 4 valmis aastal 2000. Ühest küljest jätkati sellega Intel x86 protsessorite rea arhitektuuri, teisest küljest on selle mikroarhitektuuris tehtud olulisi uuendusi võrreldes P6 mikroarhitektuuriga ja nimetatud seetõttu NetBurst mikroarhitektuuriks. Selle üks olulisemaid eeliseid on süsteemsiini ja ALU kõrgendatud taktisagedused, kusjuures ALU takteeritakse protsessorist kaks korda kõrgema sagedusega. Pentium 4 uuemates mudelites on kasutusele võetud multithreading (Inteli terminoloogias hyperthreading).

Pentium 4 iseloomustavad juba varasemates protsessorites kasutusele võetud jõudluse tõstmise meetodid, nagu:

- Protsessorisisene Harvardi arhitektuur, milles cachei esimesel tasemel on käskude ja andmete vood eraldatud (eraldi cacheid käskude ja andmete jaoks);
  Superskalaarne arhitektuur, mis võimaldab mitme käsu üheaegset täitmist paralleelsetel sooritusüksustel (execution units);
  Dünaamiline käskude ümberreastamine;
  Käskude konveiertöötlus;
  Programmi hargnevuse ennustamine;

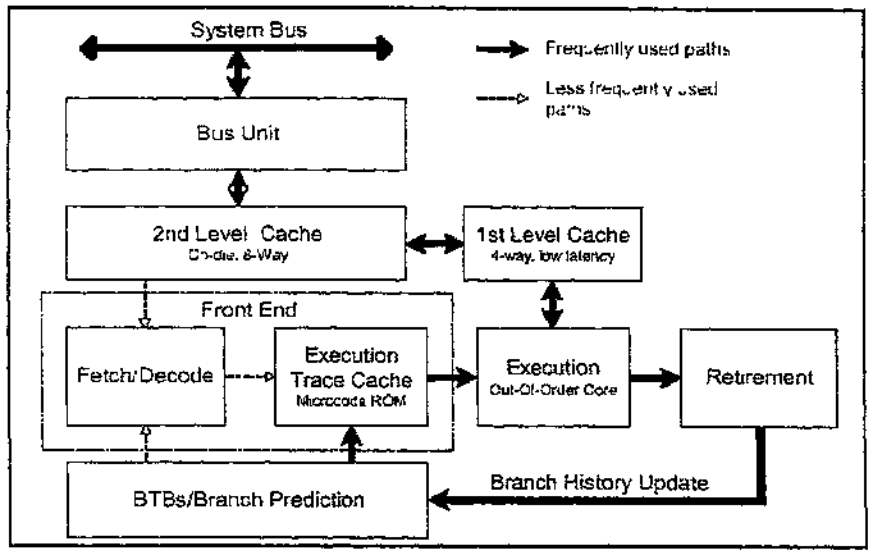
Pentium 4 lihtsustatud plokkskeem on kujutatud joonisel 5.12 , milles on kasutatud järgmisi uuendusi:

- Kahekordne ALU taktisagedus võrreldes protsessori taktiga;
- Käskude töötluse konveieri pikendamine kuni 20 astmeni;
- 144 uut SIMD (Single Instruction Multiple Data) SSE2 (streaming SIMD Extension) käsku;
- Käskude esimese taseme L1 cacheina kasutatakse nn. trace cachei, kuhu salvestatakse dekodeeritud käsud (Inteli terminoloogia järgi mikrokäsud);

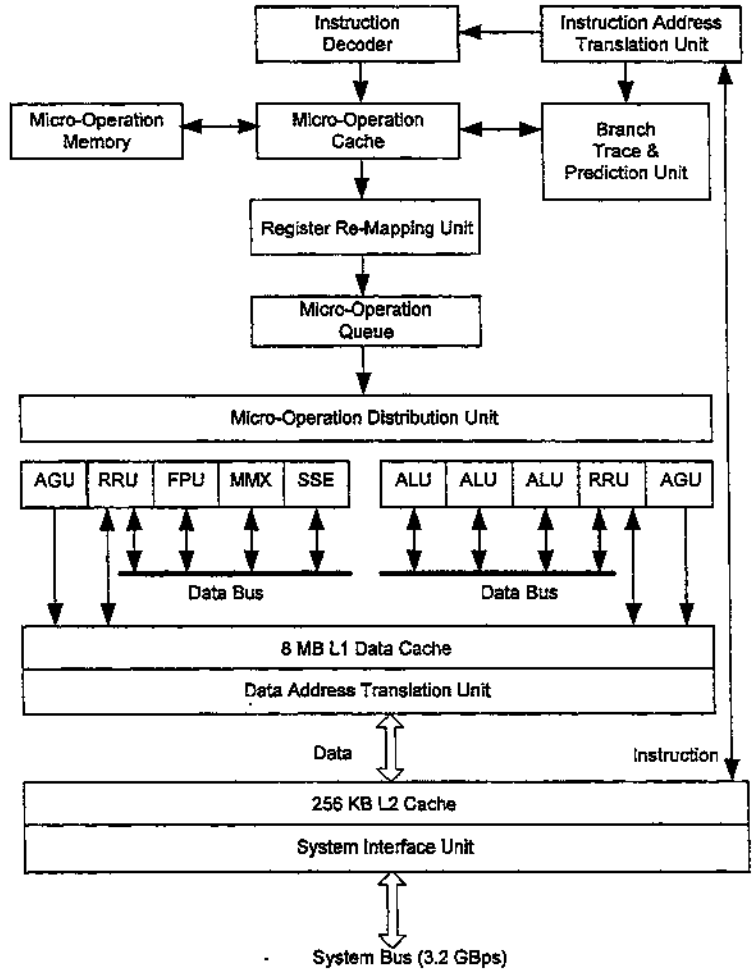
Pentium 4 struktuurskeem on toodud joonisel 5.13. L2 cache on plokk-assotsiatiivne 8-suunaline tagasisalvestusega cache . Käskude dekooder koos mikroprogrammi mäluga muudab IA käsud mikrokäskude jadadeks, mis seejärel salvestatakse trace cachei, mille maht on 12000 mikrokäsku. Mikrokäsud on cacheis järjestatud nende täitmise järjekorras, võttes seejuures arvesse ennustatud hargnemisi.

Registri määramise plokk omistab igale mikrokäskudes näidatud loogilisele registrile ühe 128-st füüsilisest registrist registriasendusplokis (RRU- Register Replacement Unit), kõrvaldades sel teel käskudevahelised sõltuvused registritest. Genereeritud mikrokäskude jada paigutatakse järjekorda, kuhu mahub kuni 126 IA käsule vastavat mikrokäsku (kolm korda rohkem, kui Pentium III-s).

Mikrokäsuvõtu plokk (mikro-operation sequencing unit) valib järjekorrast täitmiseks mikrokäske mitte samas järjekorras, millises need sinna olid paigutatud, vaid sõltuvalt mikrokäskudele täitmiseks vajalike operandide ja sooritusüksuste kättesaadavusest. Rööbiti töötavatel sooritusüksustel saab



Joonis 5.12



Pentium 4 microprocessor structure.

Joonis 5.13

üheaegselt mittejärjekorras täita kuni kuus mikrokäsku. Mikrokäskude täitmise tulemused salvestatakse põhimällu käskude programmis paiknemise järjekorras. Käskude ümberjärjestamine toimub ümberjärjestuse puhvris samuti kui eelmistes Inteli protsessorites.
Mälust loetavate operandide aadressid arvutatakse aadressigenereerimise plokis (AGU- Adress Generation Unit), mis realiseerib liidese 8 Kbaidise tagasisalvestusreziimis töötava L1 cacheiga.
AGU genereerib käskude jaoks , mis pole veel võetud täitmiseks, 48 aadressi mälust RRU registritesse laadimiseks ja 24 aadressi registritest mällu salvestamiseks. Mällu pöördumisel genereerib AGU kaks aadressi: ühe operandi laadimiseks mälust RRU registrisse, teise tulemuse salvestamiseks RRU registrist mällu.
Protsssori superskalaarne tuum sisaldab konveierreziimis töötavaid tehete sooritamise plokke: kolm ALU-d täisarvulise aritmeetika jaoks, FPU ujukoma aritmeetika jaoks, MMX (MultiMedia Extension) multimeedia arvutuste jaoks. MMX sooritab ühe käsuga tehte üheaegselt mitme operandiga (SIMD-Single Instruction Multiple Data). Pentium 4 hüperkonveier (hyperpipeline) koosneb 20-st astmest. Tänu sellele, et käsutäitmise tsükkel on jagatud lühemateks astmeteks, millest igaühte on võimalik kiiremini täita, on võimalik protsessori taktisagedust tõsta.
Kuid teisest küljest, suurendades konveieri astmete arvu, suurenevad ka ajakaod hargnemiskäskude täitmisel valesti valitud haru puhul. Neid kadusid saab vähendada harude võimalikult õige ennustamisega haruennustuse plokis. Hargnemise sihtaadresside plokk, mis on osa haruennustuse plokist, säilitab kuni 4092 varem kasutatud sihtaadressi koos nende kasutamise eellooga. Pentium 4 ennustusplokk võimaldab kuni 90%- st haruennustuse edukust.
Pentium 4 omab laiendatud võimalustega SIMD plokki SSE2, mis võrreldes Pentium III vastava plokiga SSE, 144 uut SSE käsku ja võib töödelda mitut operandi , mis võivad paikneda nii mälus, kui ka 128-bitistes XMM0...XMM7 registrite. Kaks topelttäpsusega ujukomaarvu arvu (64 bitti) või neli tavatäpsusega ujukoma arvu (32 bitt) võivad olla laaditud ja üheaegselt töödeldud mainitud registrite abil. SSE2 plokk võib üheaegselt töödelda ka kinniskoma arve järgmiselt: 16 8-bitist, 8 16-bitist, nelja 32-bitist või kahte 64-bitist.
Pentim 4 omab võrreldes Pentium III-ga kahekordset jõudlust. Protsessoris on 42 miljonit 0,18- mikronilises CMOS tehnoloogias integreeritud transistorit.

6.1 RISC protsessorid

Silmas pidades käskude konveiertöötlust tekitab probleeme operandide lugemine mälust, mis nõuab ootetaktide lisamist, aeglustades sellega konveieri tööd. Operandide lugemine registritest on väga kiire ja võib toimuda samal taktil tehte sooritamisega. Seevastu mälus paiknevad operandid, isegi siis kui need resideeruvad cache-is, nõuavad lugemiseks vähemalt ühte lisatakti enne, kui järgnev käsk saab neid kasutada Probleem tekib näiteks siis, kui programmis järgnevad üksteisele kaks käsku, millest esimene on registri laadimise käsk mälust ja sellele järgnev liitmise käsk:

```
LOAD    VALUE, R5
ADD     R5, R4, R3
```

Kuna mälu on aeglasem, kui protsessor ei jõua laaditav väärtus VALUE veel selleks ajaks registrisse R5, kui liitmiskäsk ADD alustab sooritamise takti. Et vältida käsu täitmist vale operandiga, tuleb konveieri tööd (aparatuurselt) ootetaktide lisamise abil ajutiselt peatada.

RISC arhitektuur püüab selle probleemi mõju minimeerida sel teel, et eraldab, niivõrd kuivõrd programmi sisu seda võimaldab, mällu pöördumise käsud (lugemine ja salvestus) tehete sooritamise käskudest eraldi gruppidesse. Selline eraldamine annab seda parema tulemuse, mida suuremaid (teatud optimaalse piirini muidugi) käskude gruppe õnnestub kompaileril moodustada ja mida rohkem operande mahub korraga protsessorisse. Viimase nõude täitmiseks peab protsessoris olema piisavalt suur arv registreid. Enamuses RISC arhitektuuriga protsessorites on 32 adresseeritavat registrit.

Tulemuste mällu salvestamine ei ole nii problemaatiline protsessori töö pidurdumise seisukohast, kuna tulemust on võimalik peaaegu alati registrisse laadida, kust see koos teiste juba registritesse laaditud tulemustega sobival ajal mällu salvestatakse.

Optimaalsed grupiviisilise registrite laadimise ja registritest mällu salvestamise ajad ja käskude arvu paneb paika kompailer programmi masinakeelde transleerimise käigus. Siit ka järeldub, et RISC arhitektuur nõuab kõrgkeeles kirjutatud programmi teksti transleerimiseks keerukamat ja seetõttu ka mahukamat kompailerit, mis kulutab kompileerimiseks ka rohkem aega. Kuid programmi kompileeritakse üks kord, täidetakse aga sadu ja tuhandeid kordi, mistõttu kompileerimisele kulutatud aega programmi paljukordsel kasutamisel arvesse võtta pole mõtet. Tähtis on, et kompileerimise tulemusena saadud masinakeeles programm töötaks võimalikult kiiresti.

RISC arhitektuuri põhilisteks tunnusteks on kompaktne käsustik, käsuvormingu lihtsus, mälu adresseerimise lihtsus, protsessori mikroarhitektuuri lihtsus, protsessori registrite suur arv ja kompilaatori keerukus, mis on tingitud lähteprogammi teksti sügavama analüüsi vajadusest .

6.2 SPARC arhitektuuriga RISC protsessorid

Scalable Protsessor ARChitecture (SPARC) - mastabeeritav protsessori arhitektuur on välja töötatud firma Sun Microsystems poolt 1985 aastal. SPARC protsessorite pere koosneb 32-bitistest protsessoritest MicroSPARC, Super SPARC, HyperSPARC ja 64-bitisest UltraSPARC. Nende protsessorite peamisteks kasutusaladeks on kõrge jõudlusega tööjaamad, serverid ja superarvutid. SPAC arhitektuur põhineb RISC I ja RISC II uuringutel, mis viidi läbi USA Kalifornia Berkley ülikooli juhtimisel aastatel 1980 kuni 1982.a.

SPARC arhitektuuri peamisteks tunnusteks on:

- 32- bitisel aadressil põhinev lineaarne aadressiruum 2 = 4 gbaiti;
- fikseeritud struktuuriga ja võrdse pikkusega (32 bitti) kolm käskude põhiformaati;
- mälu ja sisend/ väljundseadmete poole pöördumine toimub laadmis/salvestuskäskudega;
- kasutab kolmeaadressilisi register- adresseerimisega käskusid;
- suur registrifail, mis kasutab registeraknaid (register windows); igal ajahetkel on programmil kasutada 8 globaalregistrit ja 24-registriline registeraken, mis on kaardistatud registrifaili. Registerakende kasutamine võimaldab oluliselt vähendada ajakadusid, mis on seotud töökeskonna ümberlülimisega paralleelprotsesside täitmisel ja protseduuridesse pöördumisel;
- eraldi ujukoma registrifail; tarkvara võib interpreteerida seda registriteplokki, kui 32-te ühekordse täpsusega 32-bitist registrit, 16-nd topelttäpsusega 64-bitist registrit, 8-t neljakordse täpsusega 128-bitist registrit või läbisegi erineva täpsusega registreid;

SPARC arhitektuur kasutab järgmisi andmetüüpe ja formaate:

- märgiga täisarvud: 8, 16, 32 ja 64 bitti;
- märgita täisarvud: 8, 16, 32 ja 64 bitti;
- reaal- ehk ujukomaarvud: 32, 64 ja 128 bitti;

SPARC protsessoriarhitektuuri väljatöötajana loovutas firma Sun litsentsi protsessorite tootmiseks mitmele firmale, nende seas Texas Insrumentsile, Philipsile, Fujitsule jt. Esimene SPARC protsessor

valmistati 1986.a. Fujitsu ja selle baasil ehitati esimene tööjaam Sun-4 1987. aastal.

6.3 Ultra SPARC arhitektuur.

Uus protsessorite põlvkond suurendas oluliselt SPARC süsteemide graafika- ja videotöötluse võimalusi. UltraSPARC, mille plokkskeem on toodud joonisel 6.1 on üks esimesi unversaalprotsessoreid, mis reliseerib RGB formaadis graafika- ja videoandmete töötlust riistvaras. Selle protsessori spetsiaalne moodul töötleb kaheksat kujutiseelementi üheaegselt.

Protsessori käsustiku koosseisu kuulub 30-st käsust koosnev kujutisetöötluse pakett VIS (Visual Instruction Set), mis laadib ja töötleb andmeid 64-bitiste plokkidena. Digitaaltelevisioonis kasutatava MPEG koodeki (kooderi-dekooderi) algoritmidest on kõige aeganõudvamad liikumist analüüsivad ja jooksvat kaadrit eelmise kaadriga võrdlevad algoritmid. UltraSPARC-i spetsiaalsed käsud võimaldavad kujutisetöötluse kiirust suurendada kuni 80 korda, võrreldes teiste SPARC protsessoritega.

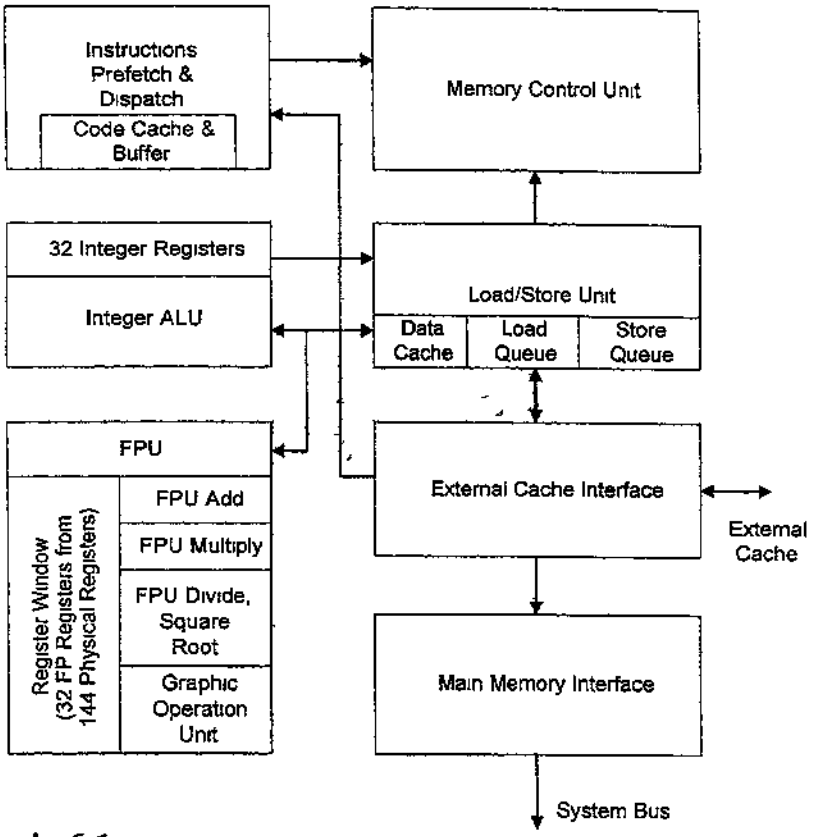
UltraSPARC-i käskude konveier on üheksaastmeline ja võimaldab täita kuni neli käsku igal taktil. Protsessor ei muuda käskude täitmise järjekorda. Igal taktil võib protsessor täita kaks tehet täisarvudega, kaks ujukomaarvudega, ühe laadimis/salvestuskäsu ja ühe hargnemiskäsu. Olgugi et kokku on võimalik täita üheaegselt kuus käsku, reaalselt õnnestub siiski täita vaid neli. Protsessor täidab käskusid küll järjestikku, kuid tulemused ei pruugi olla esitatud samas järjekorras kui täidetavad käsud.

UltraSPARC III on Sun Microsystemsi V9 arhitektuuriga kolmanda põlvkonna protsessor. Erinevalt eelmistest SPARC perekonna protsessori põlvkondadest, mis nõudsid arhitektuuri spetsiifika arvessevõttu operatsioonsüsteemi tasemel, ühildub see protsessor kõigi operatsioonsüsteemidega ja rakendustega, mis on välja töötatud SPARC protsessoritele.

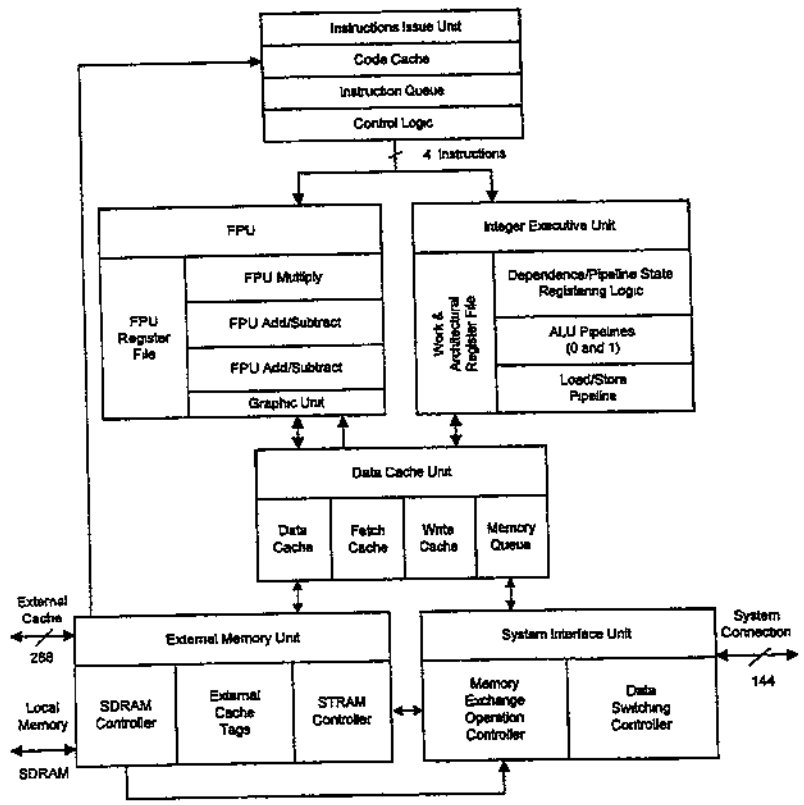
Sarnaselt eelmiste põlvkondadega on UltraSPARC III-s rakendatud haru ennustust, käskude tasemel parallelismi (Instruction- level Parallelism-ILP) avastamist kompileerimise tasemel ja registeraknaid (Register Windows). Käskude täitmise konveier on 14-astmeline.

Arhitektuursed registrid ja tööregistrid on protsesoris üksteisest eraldatud. Käskude mittejärjekorras täitmise tulemused salvestatakse tööregistritesse ja vajadusel võivad olla sealt kas tühistatud või ümbersalvestatud arhitektuursetesse registritesse.

Protsessori UltraSPARC III plokkskeem on toodud joonisel 6.2 ja see koosneb kuuest funktsionaalplokist.



Joonis 6.1 UltraSPARC microprocessor architecture

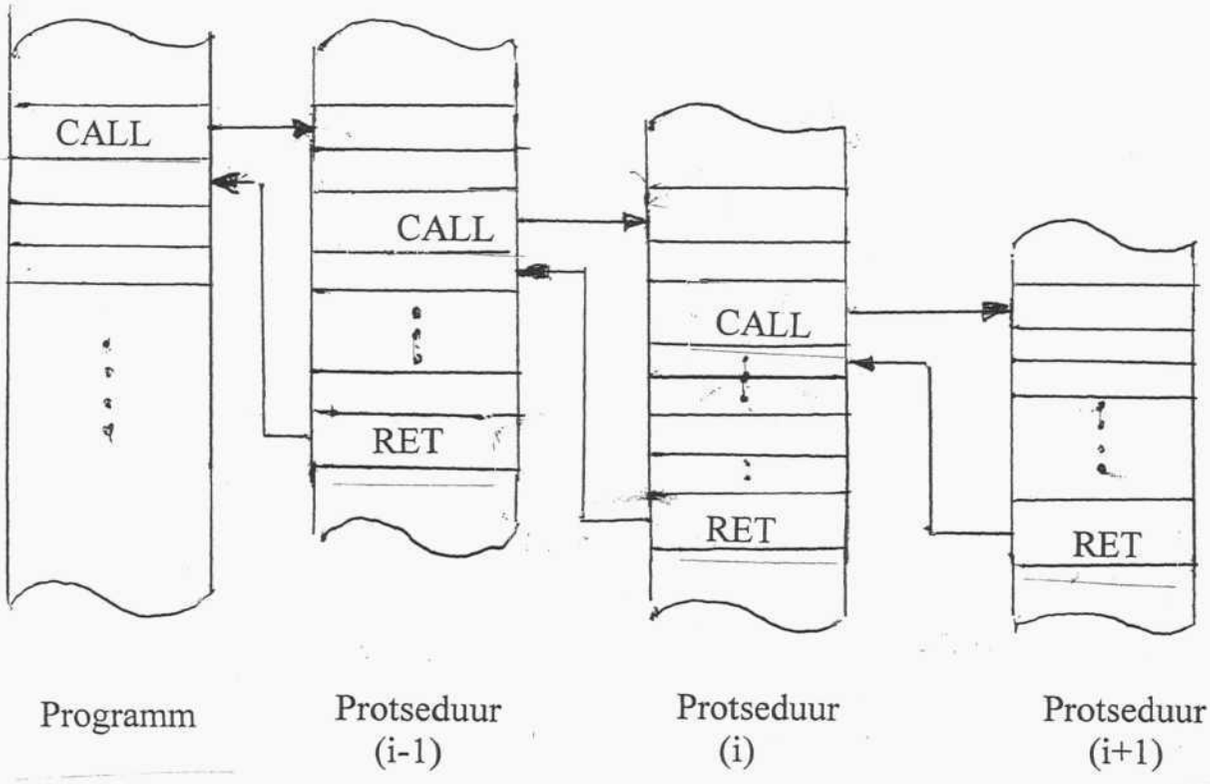


Joonis 6.2 UltraSPARC III microprocessor architecture.

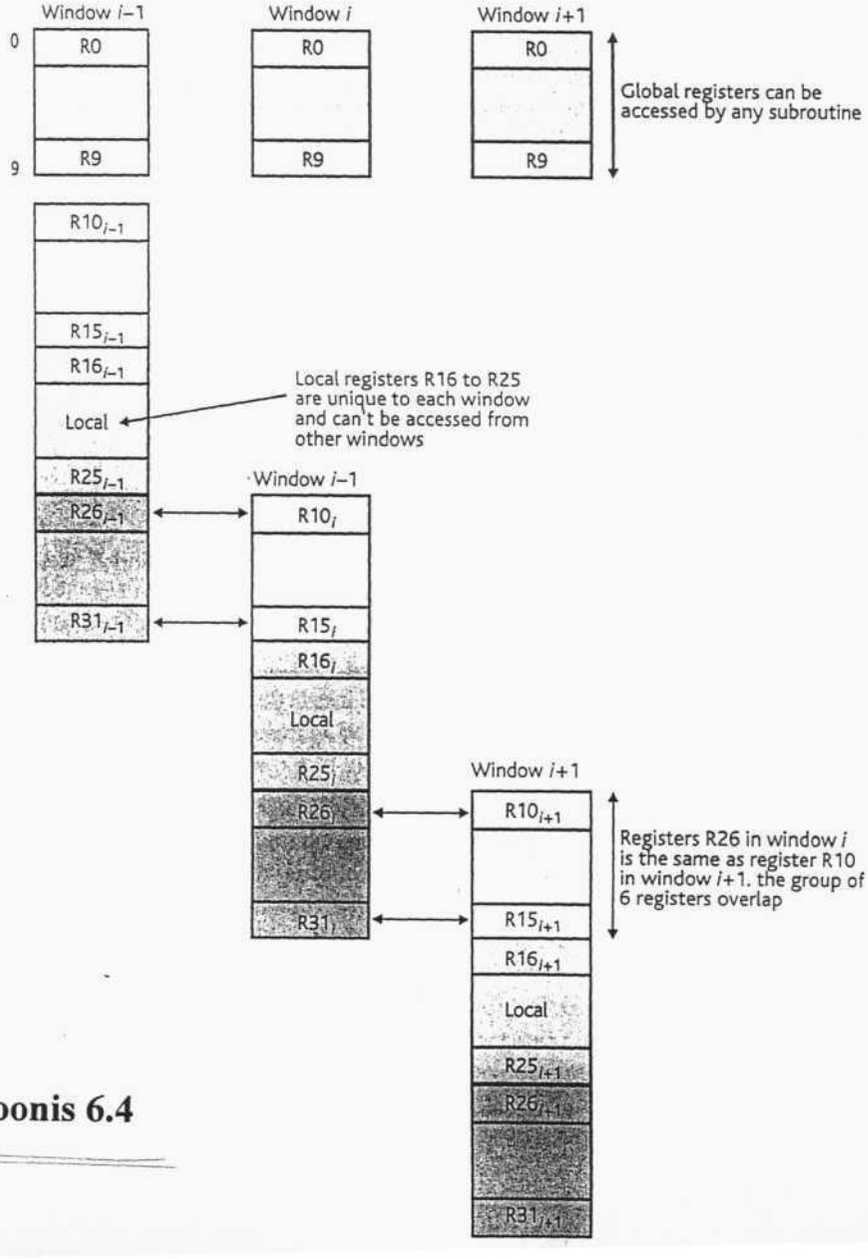
Käsuvõtuplokk (Instruction Issue Unit – IIU) ennustab hargnevusi ja valib täitmiseks käskusid, võttes arvesse ennustatud haru. Valitud käsud paigutatakse järjekorda kahte funktsionaalplokki: IEU-sse ja FPU-sse.
IIU on varustatud 4-suunalise plokk-assotsiatiivse 32 KB-se käskude cache-iga, TLB-ga (Translation Lookaside Buffer) ning 16 KB-se hargnemise ennustuse tabeliga.
Täisarvulise aritmeetika plokk (Integer Executive Unit – IEU) täidab kõiki täisarvudega sooritatavaid tehteid: andmete laadimist ja salvestamist, aritmeetika ja loogikatehteid, nihutamise ja hargnemise tehteid. Plokk võtab korraga vastu kuni neli käsku ja võib täita ühel taktil kuni neli tehet täisarvudega.
Ujukoma aritmeetika plokk (Floating Point Unit-- FPU) täidab kõiki ujukoma käske ja mõningaid käske täisarvudega (graafika käske SPARC käsustiku VIS laiendist). FPU võib üheaegselt täita kuni kolm käsku.
IEU ja FPU võivad koos täita üheaegselt kuni kuus käsku.
Andmecache-i plokis (Data Cache Unit – DCU) on andmeaadresside teisenduse puhver ja kolm cache-i : 64-KB-ne L1 andmecache, 2 KB-ne andmete eelvõtu (prefetch) cache ja 2 KB-ne andmete salvestuse cache.
Põhimällu salvestatavad andmed paigutatkse kõigepealt DCU järjekorda ja seejärel andmesalvestuse cache-i.
Protsessorivälise mälu juhtplokk juhib L2 andmecachei ja süsteemmälu SDRAM. Põhimälu kontroller toetab kuni 4 mälupanka kogumahuga 4GB.

6.4 Registeraknad (register windows)

SARC protsessorites kasutatakse protseduuridesse (alamprogrammidesse) pöördumisel registrite ümbernimetamist (ümberjaotamist). Selleks kasutatakse üksteisega osaliselt kattuvaid nn. registeraknaid (overlapping register windows). Registerakende realiseerimiseks on protsessori aparatuursete registrite arv võetud palju suurem, kui see programmeerijale paistab.
Tehtud analüüs näitab, et programmi täitmisel protseduuridesse pöördumine ja sealt naasmine kulutab palju aega, kui seda tehakse põhimälus paikneva stacki vahendusel. Protseduuride kasutamine on üldjuhul mitmetasemeline, mis tähendab et ükskõik millisest protseduurist võib omakorda pöörduda protseduuri ja sellest omakorda järgmisse jne. (joonis 6. 3).
Selleks, et protseduuride täitmine ja nendevaheline parameetrite vahetamine (saatmine ja vastuvõtt) toimuksid võimalikult kiiresti, on võetud kasutusele protseduuritasemete suurima võimaliku arvuga võrdne arv registriplokke (aknaid). SPARC protsessoris kattuvad naaberaknad üksteisega kuue parameetreid vahetava registri ulatuses (joonis 6. 4).



Joonis 6.3



Joonis 6.4

Aken on antud hetkel töötava protseduuri poolt adresseeritav 32-st registrist koosnev plokk, mis jaguneb neljaks:

R0.....R9 - 10 globaalregistrit, mida kasutavad piiranguteta kõik protseduurid;
R10...R15 – 6 registrit, mis võtavad parameetreid vastu vanem-protseduurist ja saadavad parameetreid vanemprotseduuri;
R16...R25 – 10 lokaalregistrit, mida kasutab ainult jooksvalt töötav protseduur vahepealsete tulemuste hoidmiseks;
R26...R31 - 6 registrit, mis saadavad parameetreid tütarprotseduuri ja võtavad vastu parameetreid tütarprotseduurist;

Antud akna registrid R26...R31 on ühtlasi järgmise akna registriteks R10...R15. Selleks, et saata parameetreid oma tütarprotseduuri, kirjutab antud taseme protseduur (mis on hetkel aktiivne) need oma registritesse R26....R31, kus need on ühtlasi järgmise taseme registrites R10....R15 ja on tütarprotseduuri poolt selle aktiveerimisel kohe kasutatavad.
Eriotstarbeline register, mida nimetatakse akna viidaks (window pointer – WP) osutab jooksvalt aktiivsele aknale, milleks antud juhul on aken *i*.
Pöördumine uude protseduuri toimub käsuga, mille vorming on järgmine:

CALL Rd, address

Seejuures suurendatakse akna viita 1 võrra ja programmiloenduri jooksev sisu (s.o.naasmisaadress) säilitatakse uue akna registris Rd.
Näiteks, kui protseduuris täidetakse käsk:

ADD R3, R12, R25

Siis vastab see tehtele:

R25 R3 + R12

Kus R3 asub akna globaalses aadressiruumis, R12 vanemprotseduuri väljastus- (või sisestus-) ruumis ja R25 lokaalses aadressruumis.

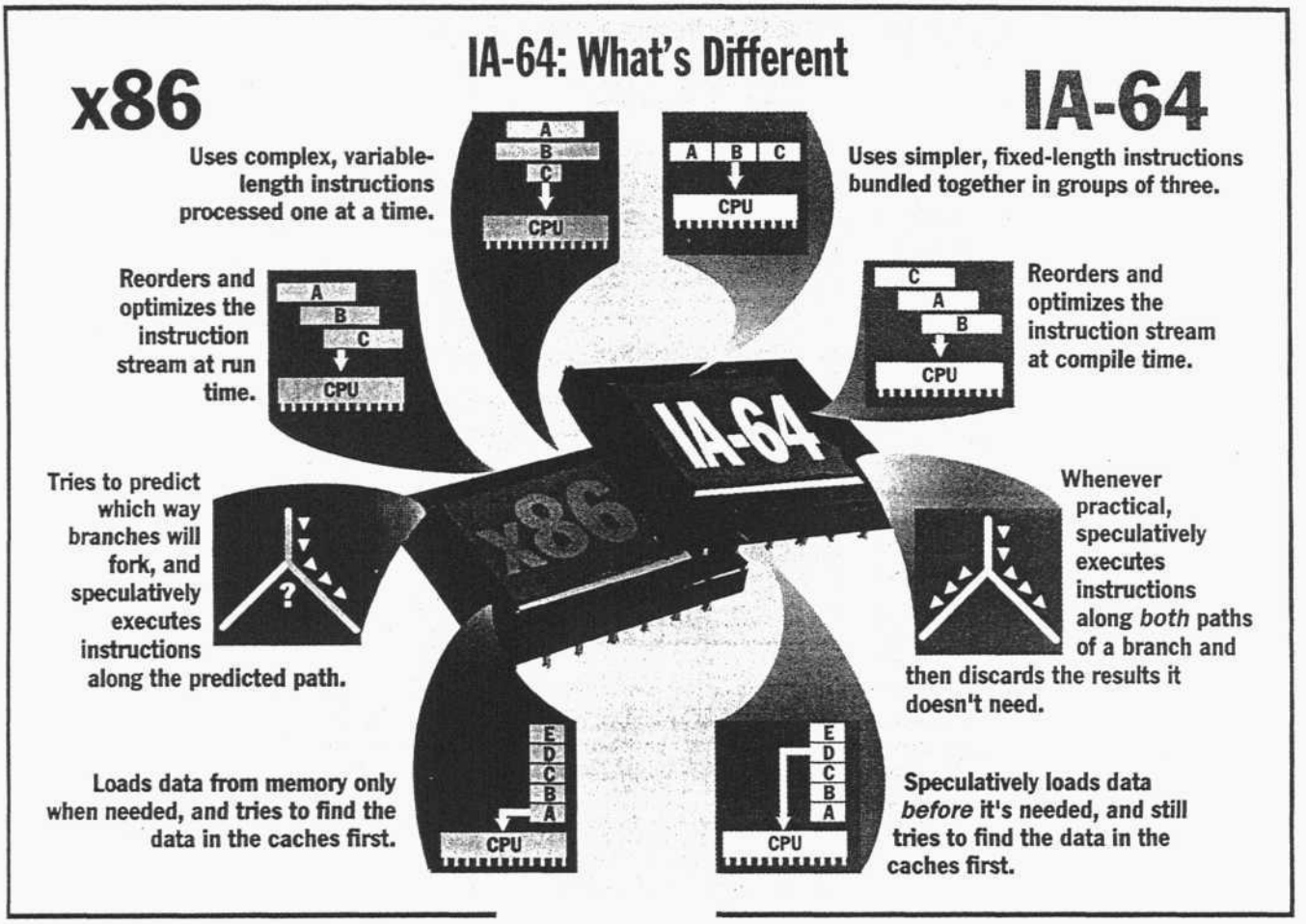
7.1 Väga pika käsusõnaga protsessori arhitektuur

Seda tüüpi protsessori arhitektuur põhineb väga pikal käsusõnal (Very Long Instruction Word – VLIW), mis on fikseeritud pikkusega ja koosneb piisavast arvust bittidest, et mahutada sellesse formaati mitu tavamõistes käsku nende üheaegseks täitmiseks. Ideaaljuhul peaks käsusõnas olema võimalik korraga esitada käsukoode ja operandide aadresse kõigi protsessori sooritusüksuste (ALU-d, FPU-d, laadimis/salvestusplokid jne.) jaoks. See võimaldaks maksimaalselt koormata kõiki paralleelselt töötavaid sooritusüksusi ja saavutada seega paralleelsest sooritusest maksimaalset tulemit. Teadagi pole ideaali kunagi võimalik saavutada, kuid selle poole pürgimisel on enim saavutusi firmade Intel ja Hewlett-Packard protsessoriarhitektuuri IA-64 väljatöötajatel, kelle ühiste pingutuste tulemusena valmis pika käsusõnaga protsessor Itanium (projekti koodnimetus Merced). Intel ei kasuta Itaniumi puhul akronüümi VLIW, vaid selle muudetud versiooni EPIC (Explicitly Parallel Instruction Computing), mis eesti keelde tõlgituna kõlaks, kui otsesõnul paralleelkäskudega arvutamine.
IA-64 põhilised erinevused oma eelkäijast IA-32-st on kujutatud joonisel 7.1 ja on järgmised: Lihtsamad, fikseeritud pikkusega, kolme kaupa grupeeritud käsud; reastab ümber ja optimeerib käskude voogu kompileerimise ajal; hargnemiskäskude puhul täidab üheaegselt käskusid mõlemas harus, õige haru selgumisel tühistab tulemused vales harus; laadib mälust andmeid spekulatiivselt enne, kui käsk neid vahetult vajab.

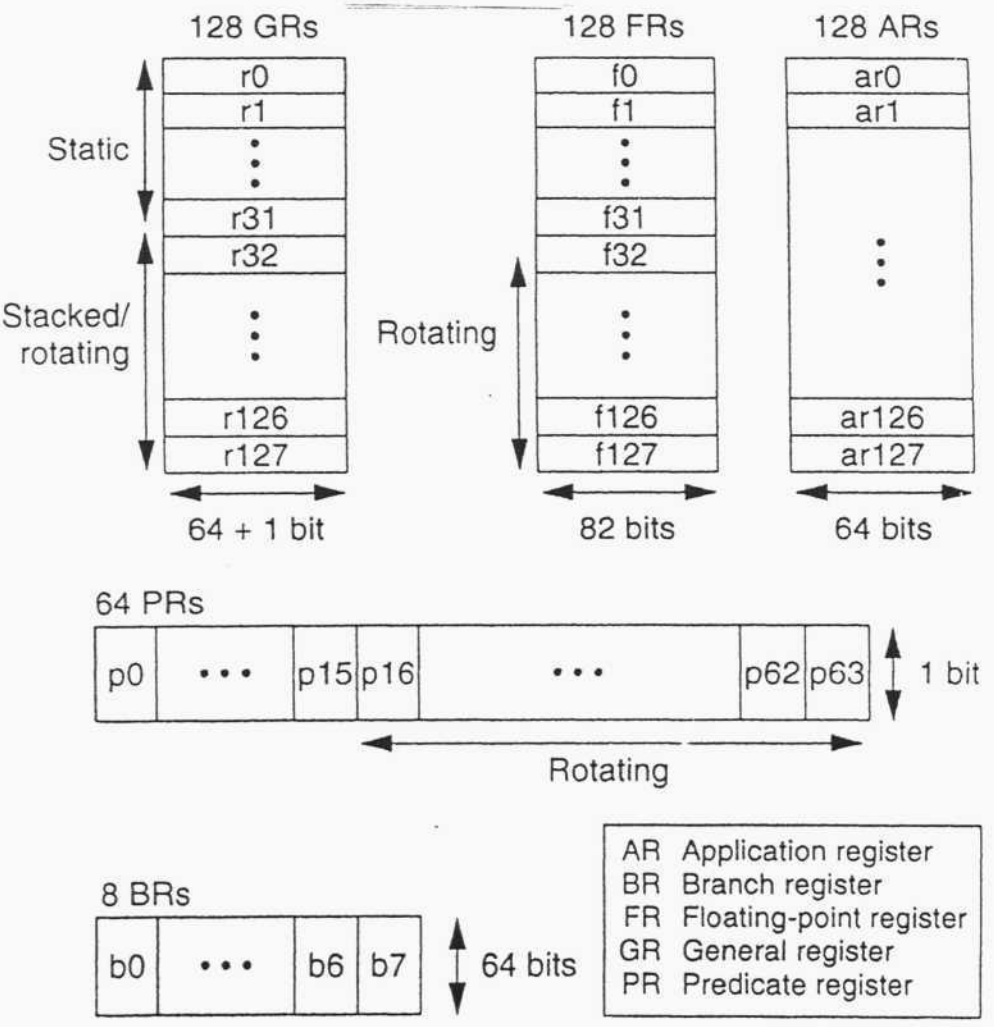
IA-64 käsustik koosneb 128-bitistest käskude pundardest (bundles), millest igaühes kolm 41-bitist RISC-tüüpi käsku. Iga pundar kätkeb endas informatsiooni käskude omavahelise sõltuvuse kohta, mis on kindlaks tehtud programmi kompileerimisel. Seda informatsiooni kasutab protsessori riistvara selleks, et planeerida puntra käskude üheaegset täitmist.
IA- 64 arhitektuuriga Itanium on esimene Inteli 64-bitine protsessor, mis tähendab et kõik selle täisarvulise aritmeetika registrid ja ALU-d on 64-bitised. Joonisel 7.2 on kujutatud Itaniumi registrid:

- 128 65 bitist (64+1) üldotstarbelist registrit andmete ja aadresside jaoks;
- 128 82-bitist ujukoma registrit;
- 128 64-bitist eriotstarbelist rakendusregistrit mida kasutatakse register- stackina ja tarkvara konveieriseerimiseks;
- 8 64-bitist hargnemisregistrit;
- 64 ühebitist predikaatide registrit;

Iga 128-bitine IA-64 pundar (joonis 7.3) sisaldab 5-bitilist maski T (template), mis paigutatakse sinna kompaileri poolt ja mis otseselt näitab protsessorile, milliseid käske saab paralleelselt täita. Edasi järgnevad



Joonis 7.1



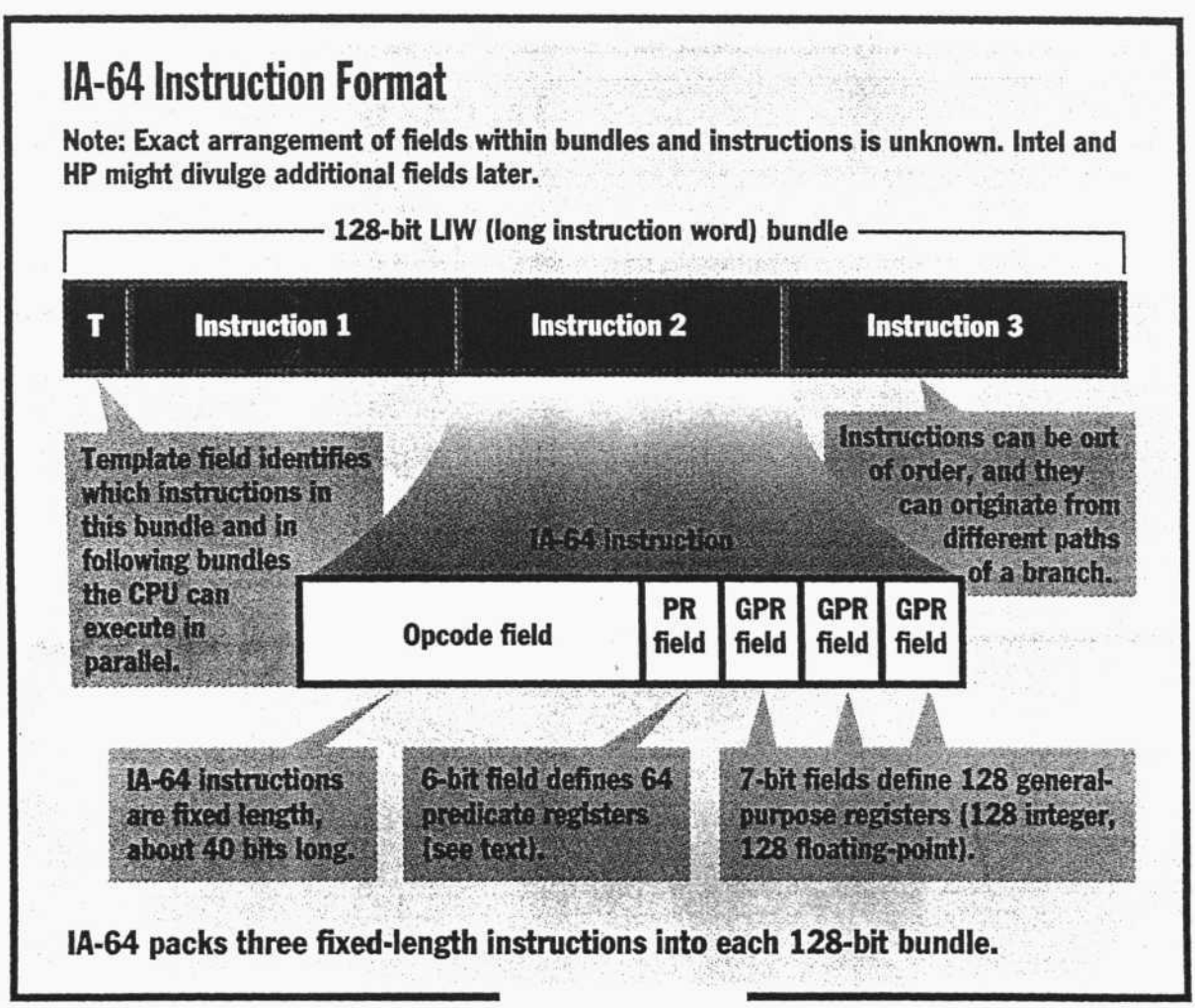
IA-64 application state. **Joonis 7.2**

kolm 41-bitist käskude välja, mis omakorda jagunevad 14-bitiseks käsukoodi väljaks, 6- bitiseks predikaadi väljaks ja kolmeks 7-bitiseks registrite aadresseerimise väljaks.
Itanium protsessori struktuur on kujutatud joonisel 2.4. Selle koosseisu kuuluvad järg,ised sooritusplokid:

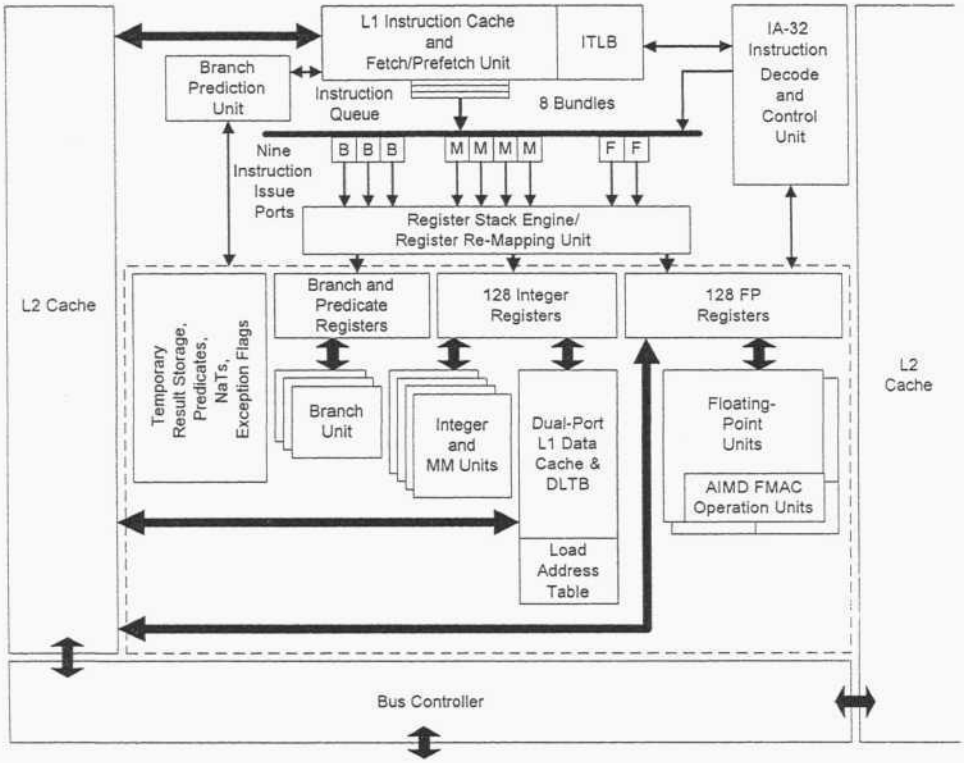
- 4 täisarvulise aritmeetika plokki;
- 4 multimeedia andmete töötlusplokki;
- 2 ühekordse täpsusega ja 2 laiendatud täpsusega ujukoma plokki;
- 2 laadimise /säilitamise plokki;
- 3 hargnemise plokki;

Kõik protsessori funktsionaalsed plokid põhinevad konveiertöötlusel. Kuni 6 käsku täidab protsessor üheaegselt. Ujukoma plokk arendab jõudlust kuni 6 GFLOPSi ühekordse täpsusega tehet ja 3 GFLOPSI laiendatud täpsusega tehet.
Protsessori otse adresseeritava süsteemmälu maht on kuni 18 GB.
Kolmetasemeline cachei hierarhia omab järgmisi näitajaid:

L1 tase koosneb eraldi andmete ja käskude cacheist, mõlemad 16 KB
L2 on ühine käskude ja andmete cache, mahuga 32 KB;
L3 on ühine käskude ja andmete kiibiväline 4 MB-ne cache, mis paikneb protsessoriga ühes kassetis;

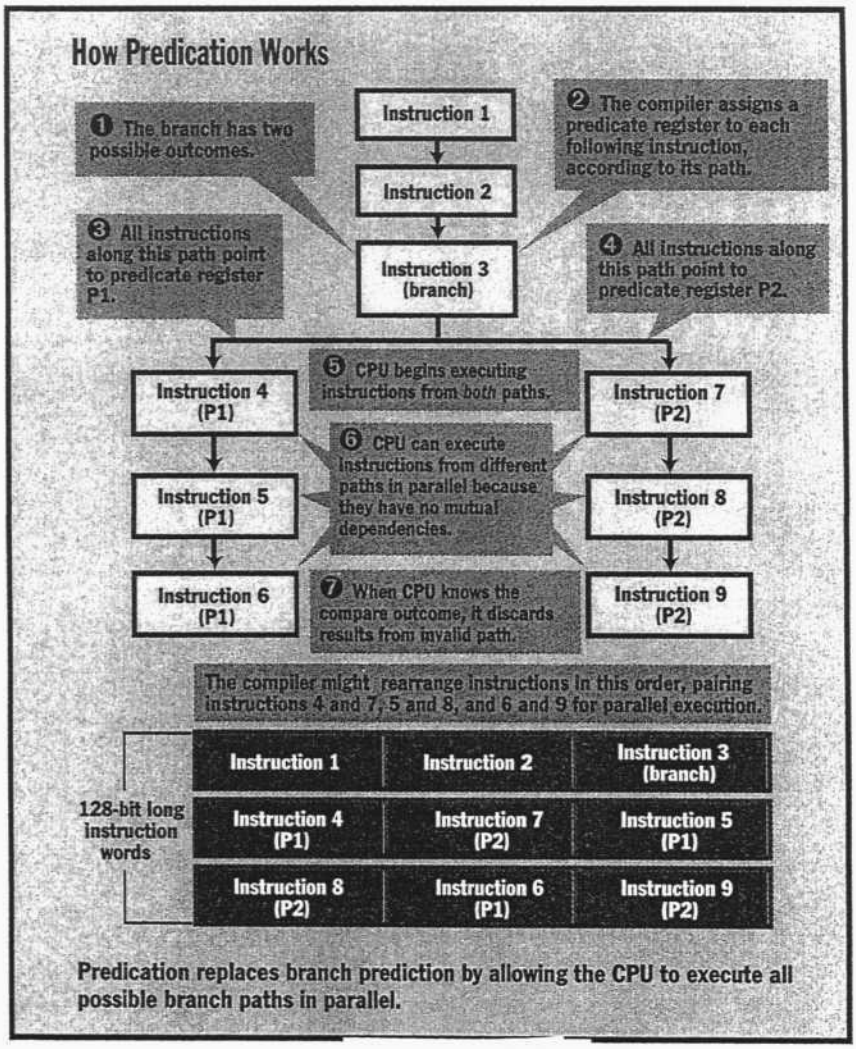


Joonis 7.3

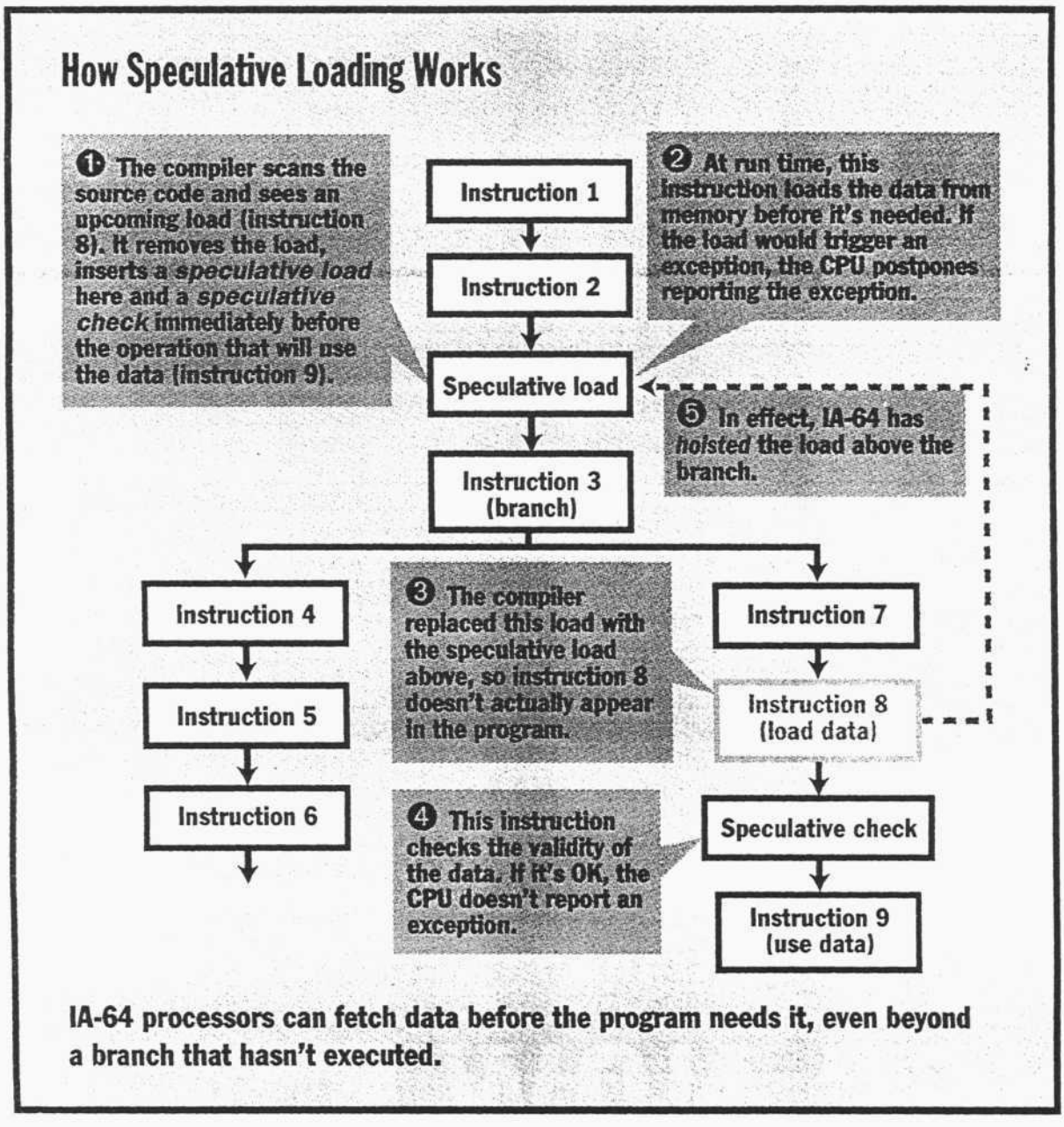


Itanium microprocessor structure block diagram.

Joonis 7.4



Joonis 7.5



Joonis 7.6

Haru ennustamine hargnemiskäsu täitmisel toimub joonisel 7.5 toodud skeemi järgi. Spekulatiivset laadimist selgitav skeem on kujutatud joonisel 7 .6.

Kirjandus

1. Korneev V., Kiselev A. Modern Microprotsessors, Third Edition, CHARLES RIVERA MEDIA, INC. Massachusetts, 2004.
2. Joseph D. Duma II, Computer Architecture, Taylor & Francis Group, 2006.
3. www.intel.com