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): :

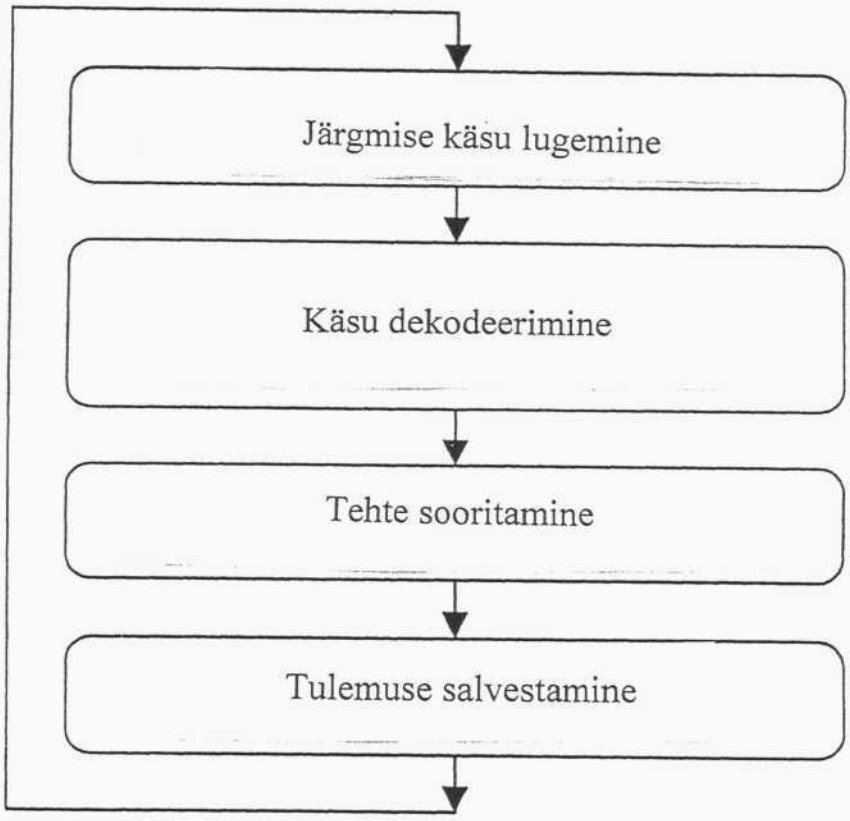
- järjekordse käsu lugemine mälupesast, mille aadress paikneb käsuloenduris. Loetud käsk fikseeritakse käsuregistris, kusjuures käsuloenduri sisu suurendatakse sõltuvalt käskude pikkusest, kas 1, 2 või 4 võrra, selleks et formeerida järgmise käsu aadress. Eelduseks on programmeerija poolt kirja pandud 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.

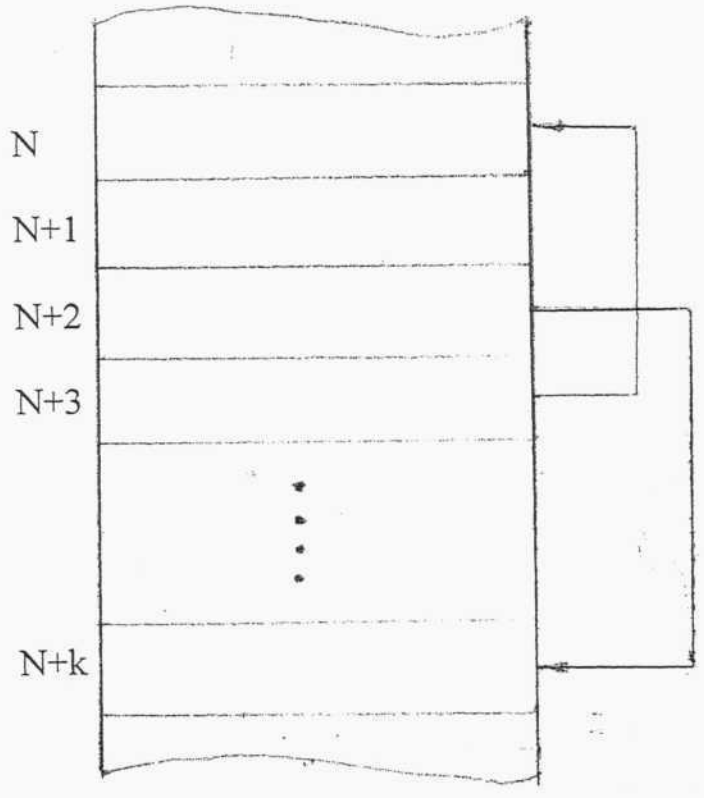
Protsessori poolt töödeldavate andmete formaat on mitmekesine. Täisarvulised operandid võivad olla ühebaidised (8 bitti), kahebaidised (16 bitti), neljabaidised (32 bitti) ja kaheksabaidised (64 bitti) või 18-kohalised kahendkodeeritud kümnendarvud (BCD). Naturaalarvulised e. ujukoma operandid omavad 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:

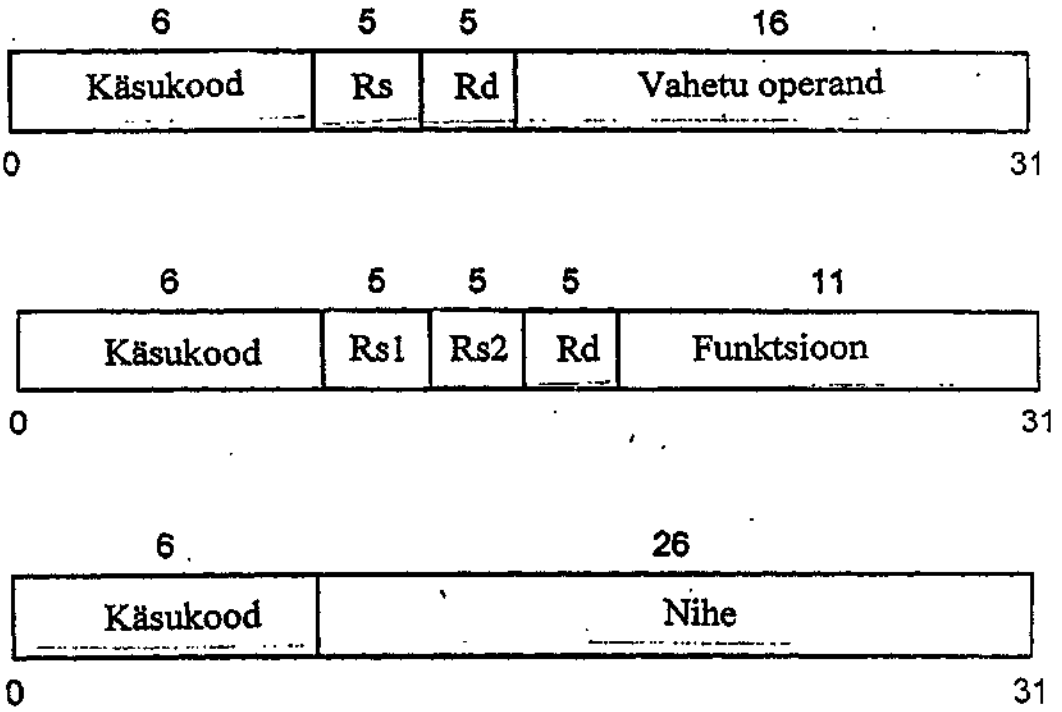
- andmeedastuskäsud siirdavad operande lähteregistrist (Rs) sihtregistrisse (Rd), vahetu operandi käsust sihtregistrisse (Rd), operande registrist mällu ja mälust registrisse;
- andmetöötluskäsud sooritavad aritmeetika-loogikatehteid;
- siirdekäsud muudavad käskude täitmise järjekorda, mis on määratud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

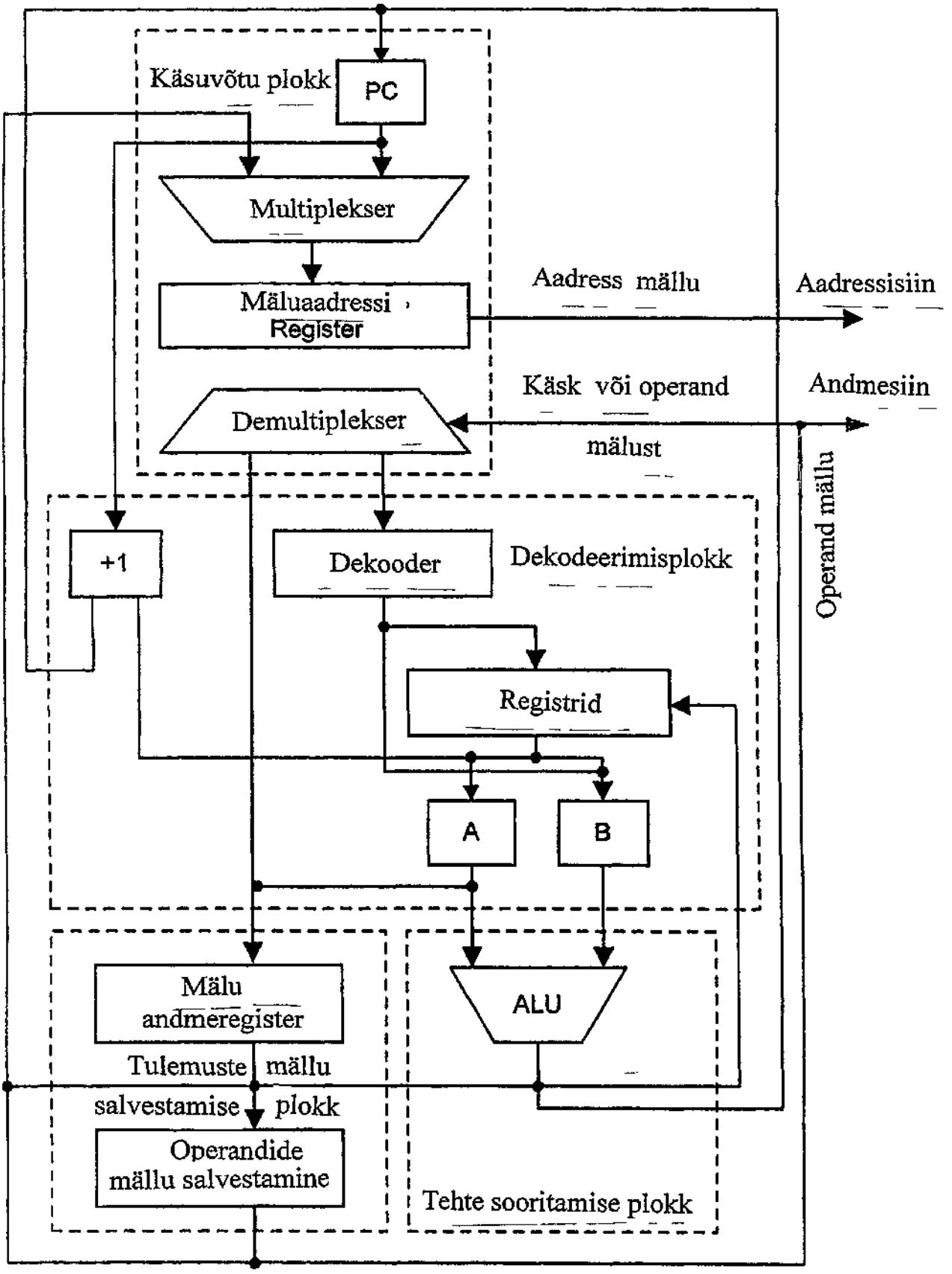
$$T = t_{kv} + t_{kd} + t_{ts} + t_{s}$$

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

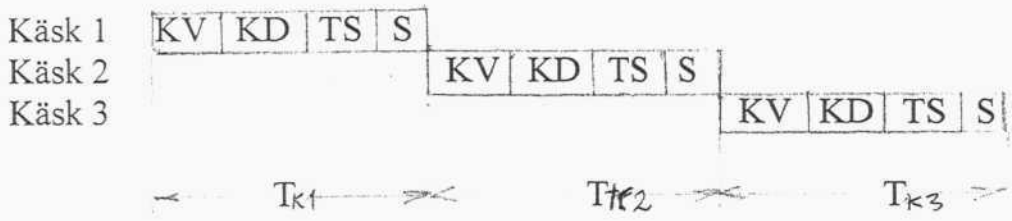
täitmine on lõpetatud (tulemus on salvestatud) (joonis 1.5). Käsu täitmise ajal on hõivatud ainult üks neljast protsessori astmest, mis viitab sellele et protsessori aparatuuri (riistvara) kasutatakse ainult ühe neljandiku ulatuses. Käsu täitmiseks kulub neli protsessori taktiperioodi: iga järgneva käsu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

1.2 Käskude konveiertöötlus.

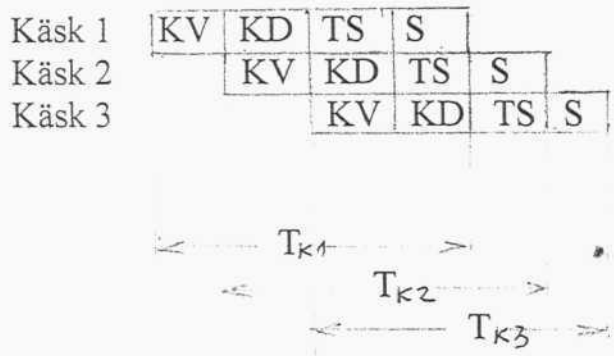
Olemasolevat aparatuurset reservi saab ära kasutada sel teel, et protsessori astmed pannakse tööle konveierina nii, nagu näidatud joonisel 1.6. Esimese käsuvõtuga mälust fikseeritakse käsk astmes 1 (takt t1). Vahetult pärast esimese käsu edastamist dekooderi astmesse 2, sooritatakse järgmine käsuvõtt 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.

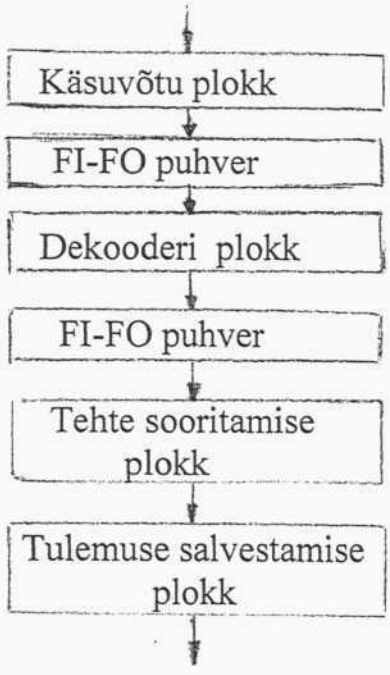
Mainitud põhjused ei piira olulisel määral konveieri rakendamisest tulenevat protsessori jõudluse tõusu. RISC- protsessorites, tänu piiratu käskude arvule ja lihtsustatud käskude vormingule, on konveiertöötluse rakendamine lihtsam ja annab paremaid tulemusi, kui CISC- protsessorites, kus suure arvu keeruka 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



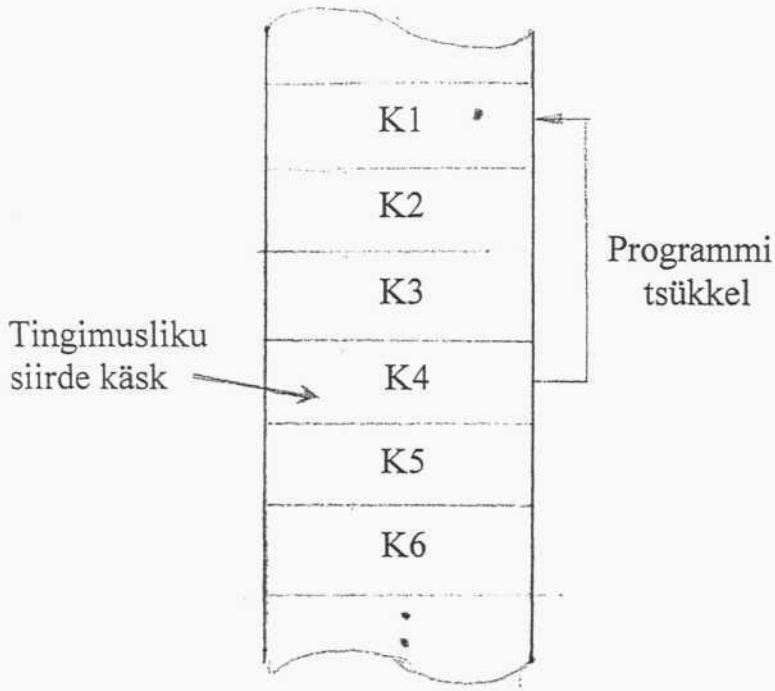
Joonis 1. 7

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.

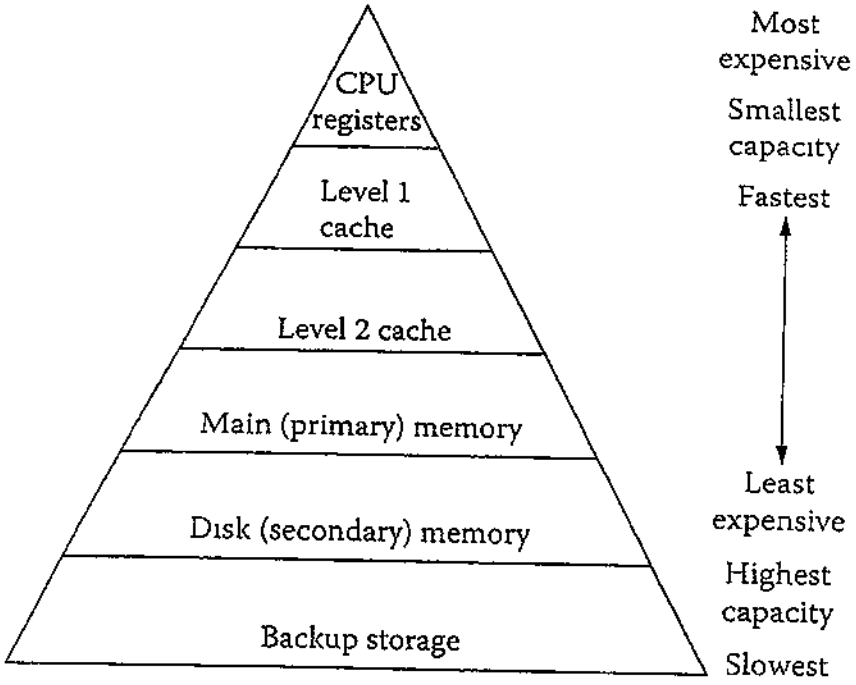
Kaks peamist arvuti mälule esitatavat nõuet on võimalikult suur andmemaht ja seejuures ka võimalikult suur kiirus (võimalikult väike reaktsiooniaeg pöördumisel). Kahjuks ei ole need nõuded seni ühelgi teadaoleval füüsikalisel põhimõttel ehitatud mäluseadmes täidetavad. Mälu andmemahu suurendamisel väheneb 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).

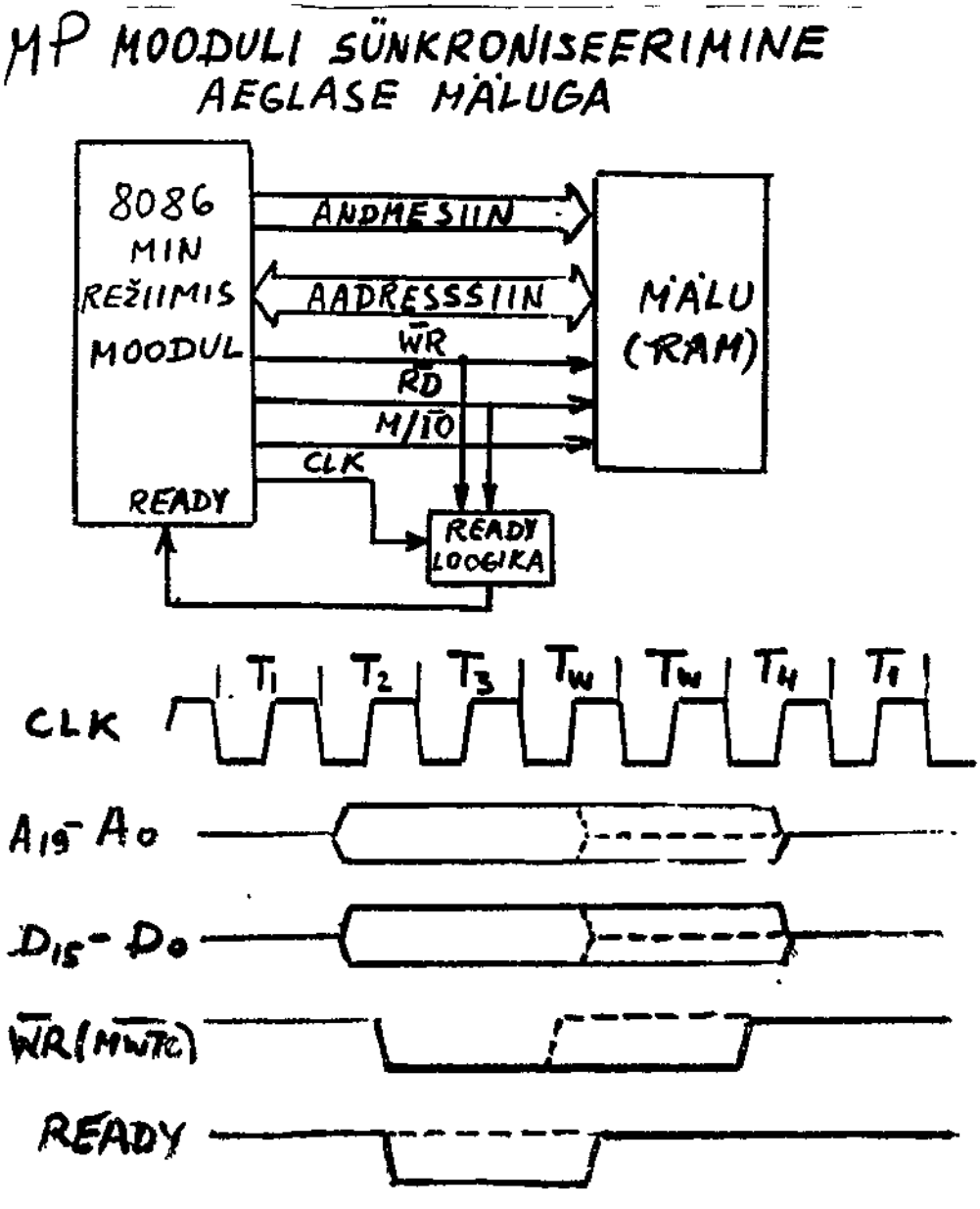
Optimeerides tasemete suhtelised andmemahud ja pöördumiskiirused on kirjeldatud mäluhierarhiat võimalik üles ehitada nii, et see moodustab protsessori jaoks piisavalt kiire pöördumisega (ilma ootetaktideta) ja praktiliselt piiramatu mahuga virtuaalse mälu. Kaasaegsetes protsessorites mahuvad protsessori 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

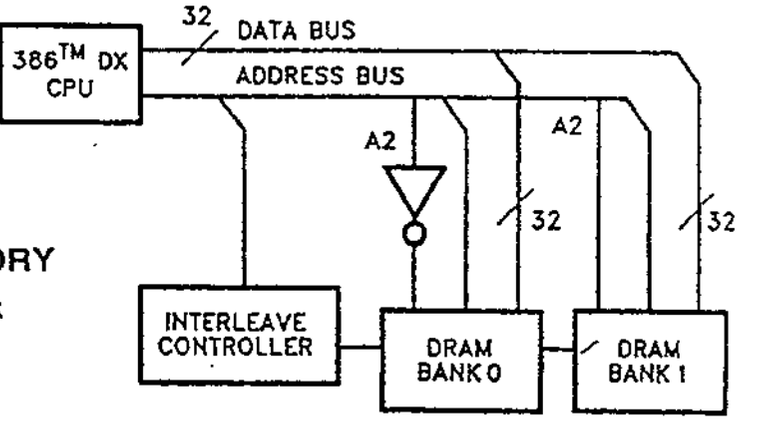


Joonis 2.2

TWO-BANK INTERLEAVED MEMORY

a) Address signal A2 selects bank

b) 32-bit datapath to each bank

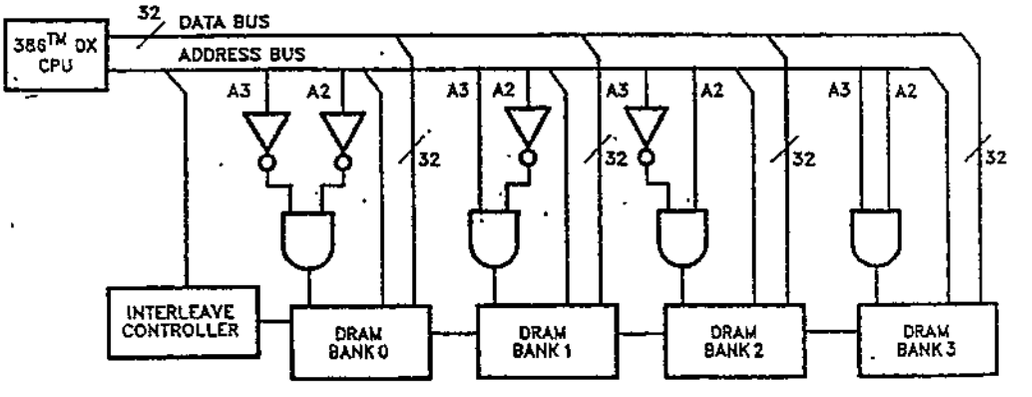


Joonis 2.3

FOUR-BANK INTERLEAVED MEMORY

a) Address signals A3 and A2 select bank

b) 32-bit datapath to each bank



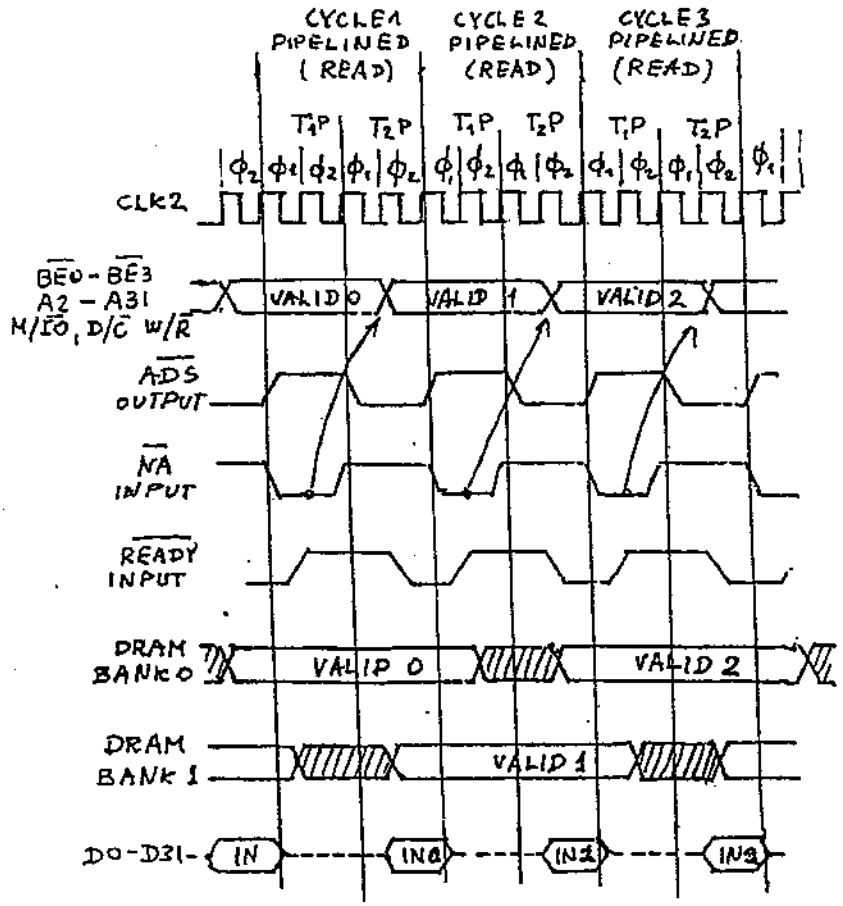
Joonis 2.6

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.

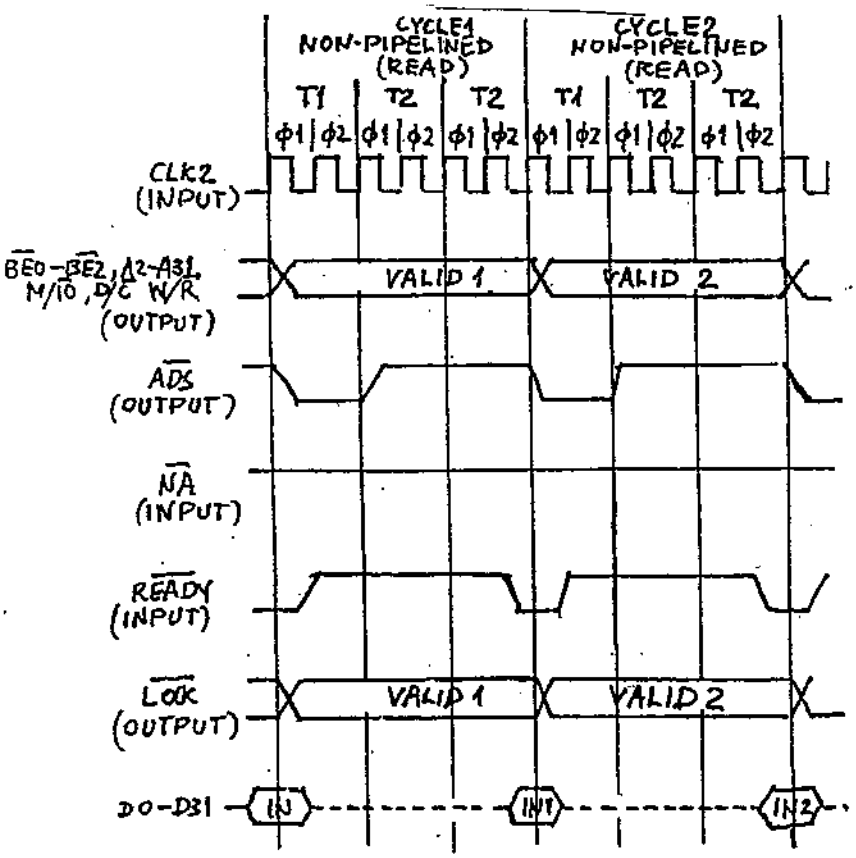
Kirjeldatud meetodit, mis põhineb vahelduval pöördumisel erinevatesse pankadesse (interleaved memory), võib laiendada näiteks neljale pangale, nagu näidatud joonisel 2.6 ning saavutada samasugust tulemust kaks korda suurema mahuga mälu puhul. Neljapangalise vahelduvpöördumisega 64 kbaidise mälu struktuur 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



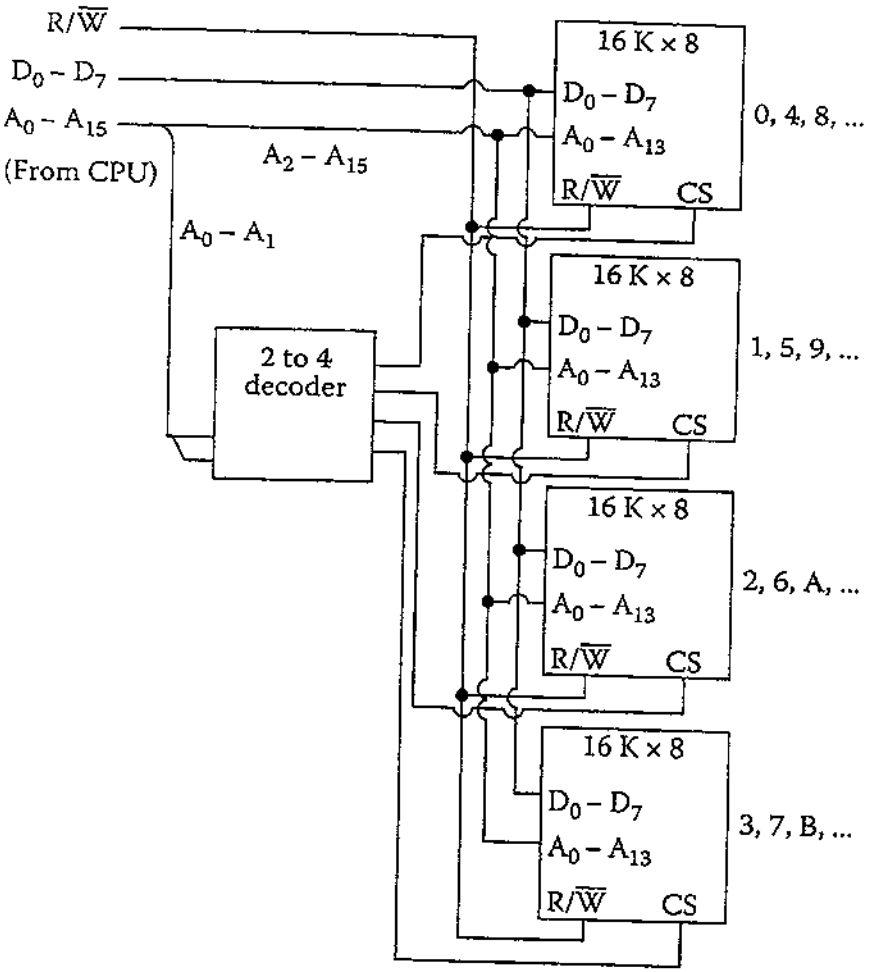
Joonis 2.4



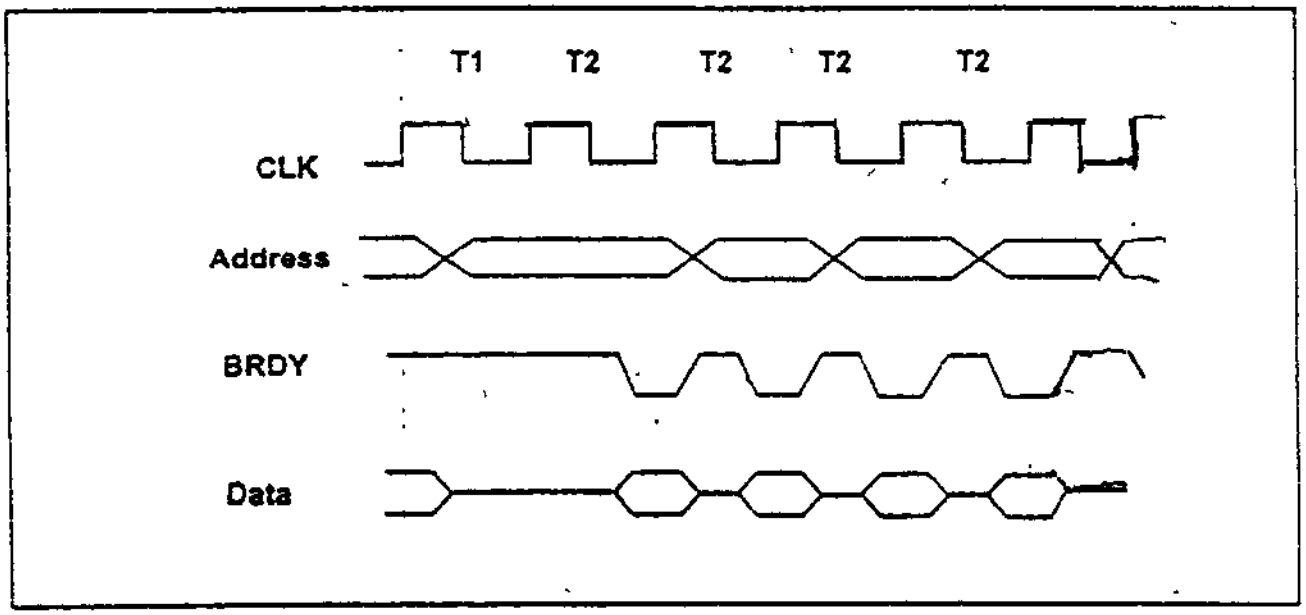
NA - NEXT ADDRESS
ADS - ADDRESS STROBE
LOCK - BUS LOCK

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.

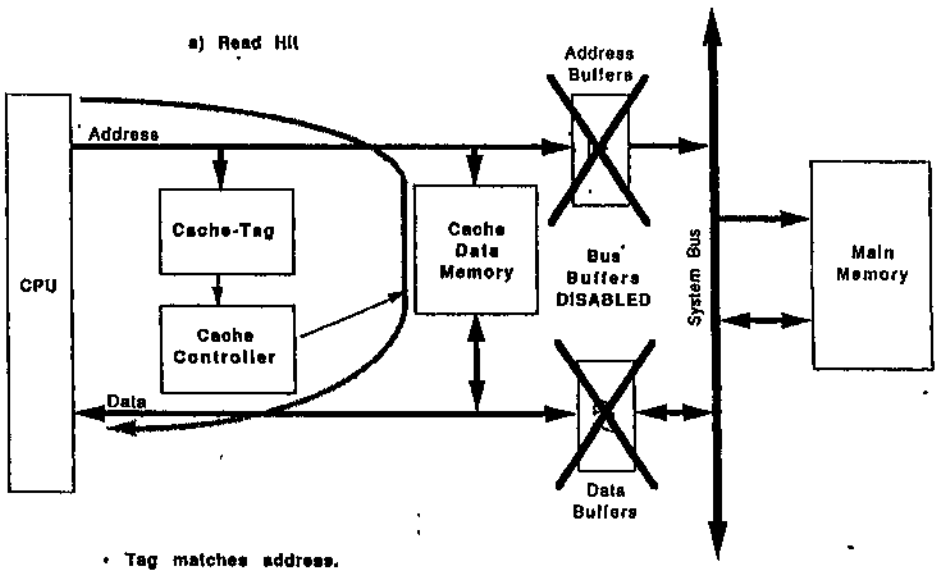
Töödeldav andmemassiiv paikneb samuti pesades, mis järgnevad üksteisele: kui mingi operand loetakse pesast *m*, siis suure tõenäosusega loetakse järgmine pesast *m+1 jne.*

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).

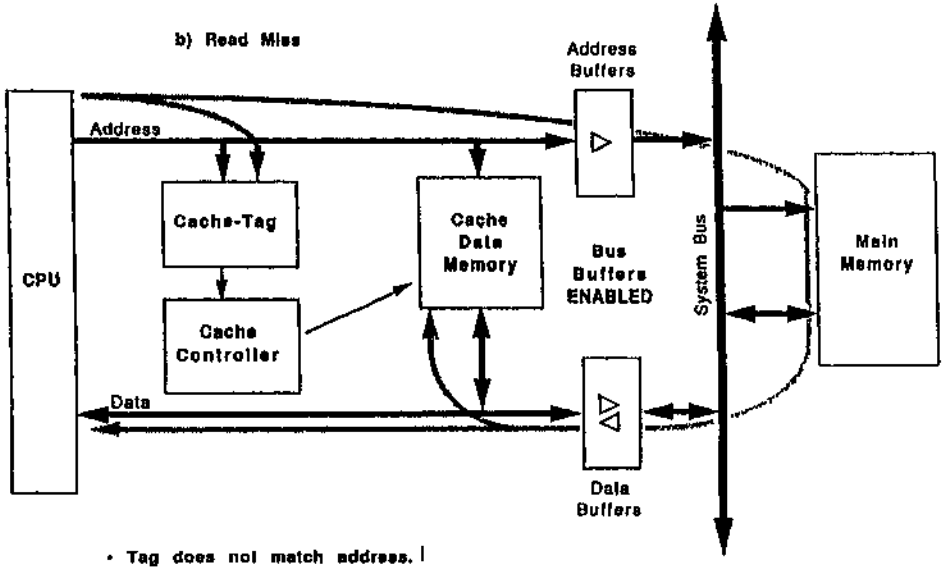
Järgmise aadressiga *n+1* mällu pöördudes leiab protsessor juba andmed cache-ist jne., kuni aadressini *n+k–1*. Edasi suunatakse pöördumine jälle põhimällu, kust loetakse cache-i rida jne. Cache-i füüsiline struktuur määrab cache-i rea pikkuse. Rea pikkus on cache-i üks olulisemaid parameetreid. Väga 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:

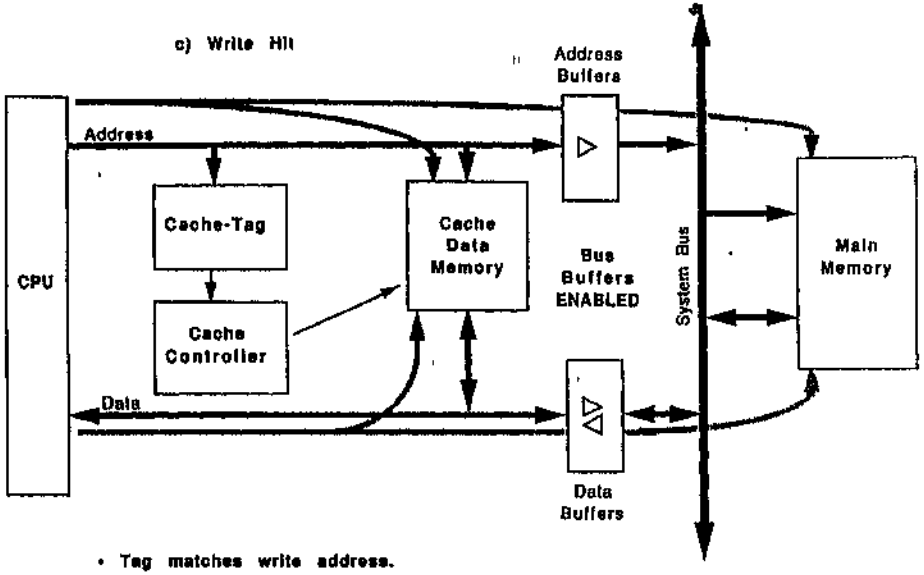
$$\frac{\text{tabamuste arv}}{\text{pöördumiste arv}} \cdot 100[\%]$$



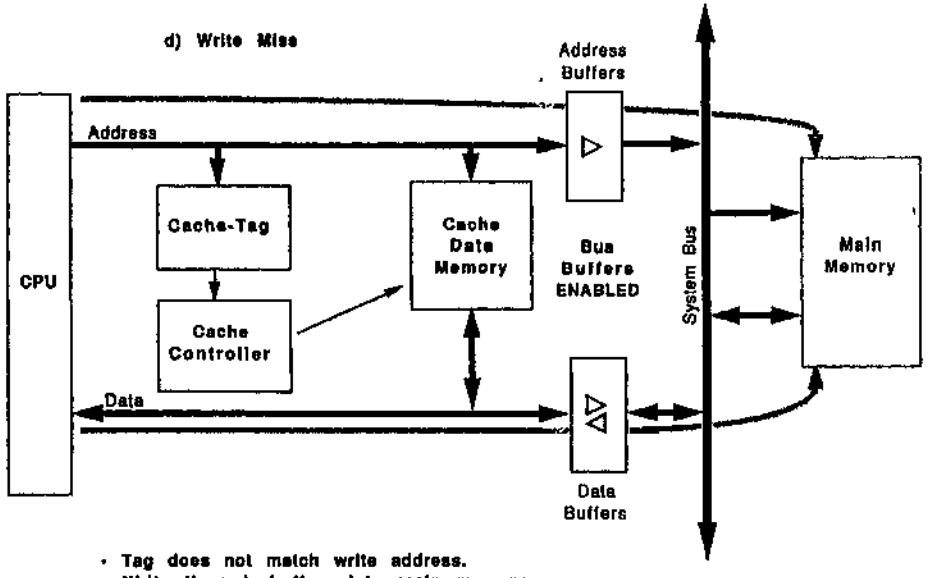
- Tag matches address.
- Read data from cache data memory to CPU.
- Disable system buffers.



- 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:

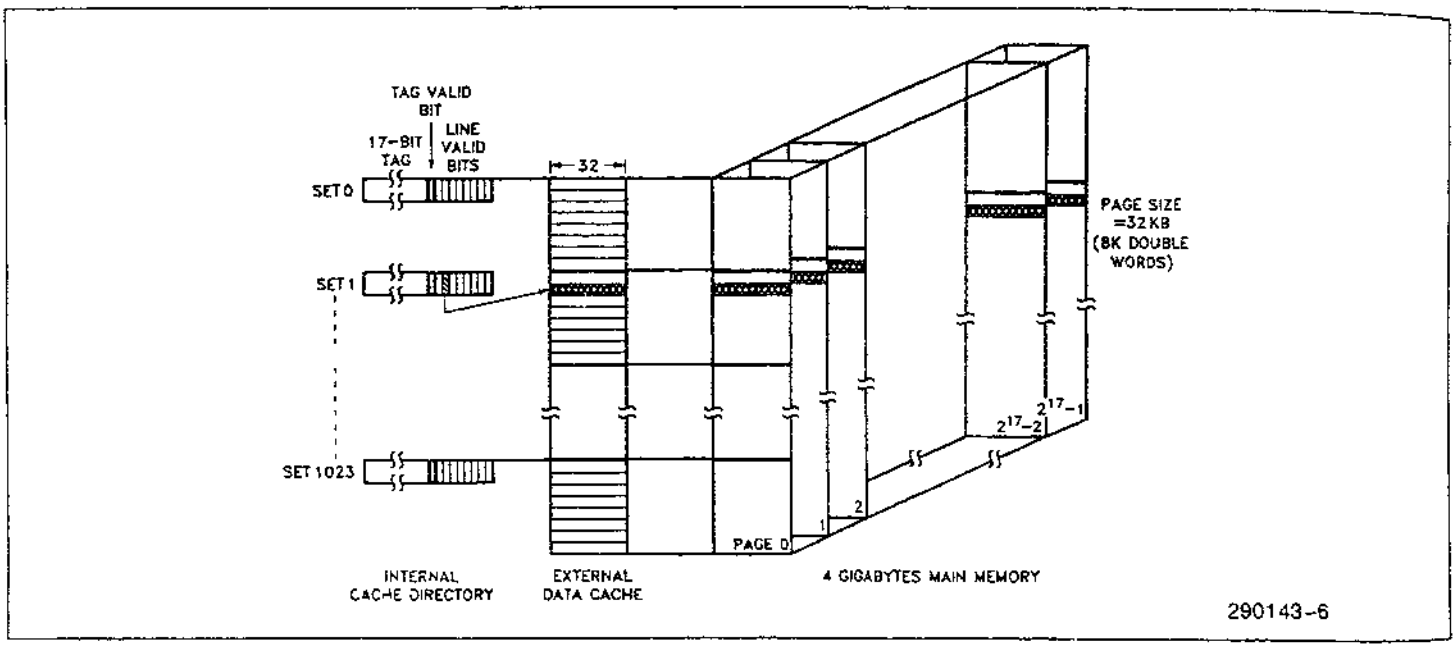
- täisassotsiatiiv (fully associative) cache;
- otsepeegeldus (direct mapped) cache;
- plokkassotsiatiiv (set assotsiative) cache;

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.

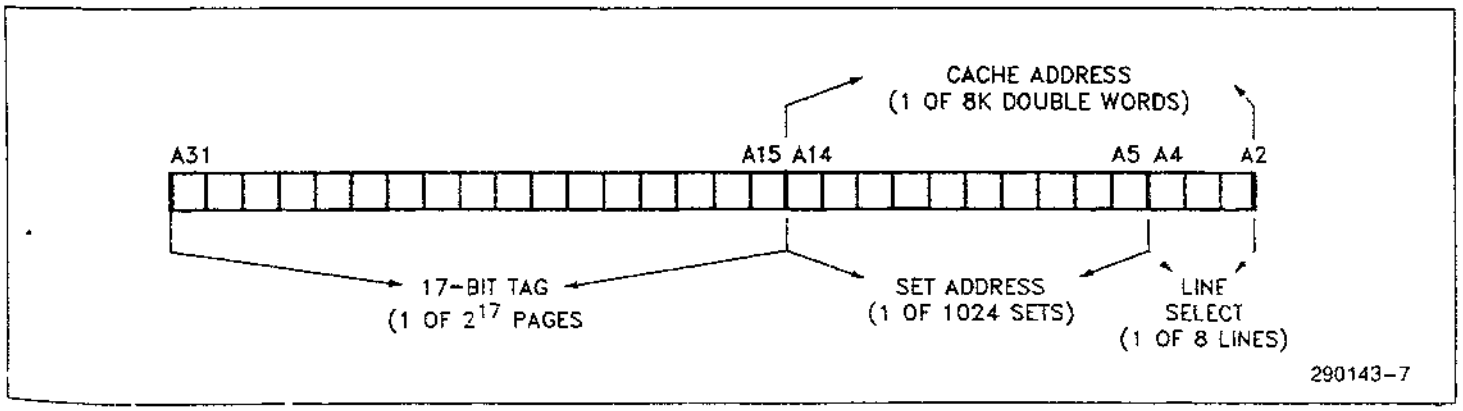
Plokkassotsiatiivcache on kas kahe-, nelja või kaheksasuunaline, mille puhul tuleb pöördumisaadressi võrrelda vastavalt kas kahe-, nelja- või kaheksa cache-is oleva aadressiga.
Nii otsepeegeldus- kui ka plokkassotsiatiivcachei üksikasjalikum tööpõhimõtte kirjeldus on toodud Intel 82385 cache-i kontrolleri 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.

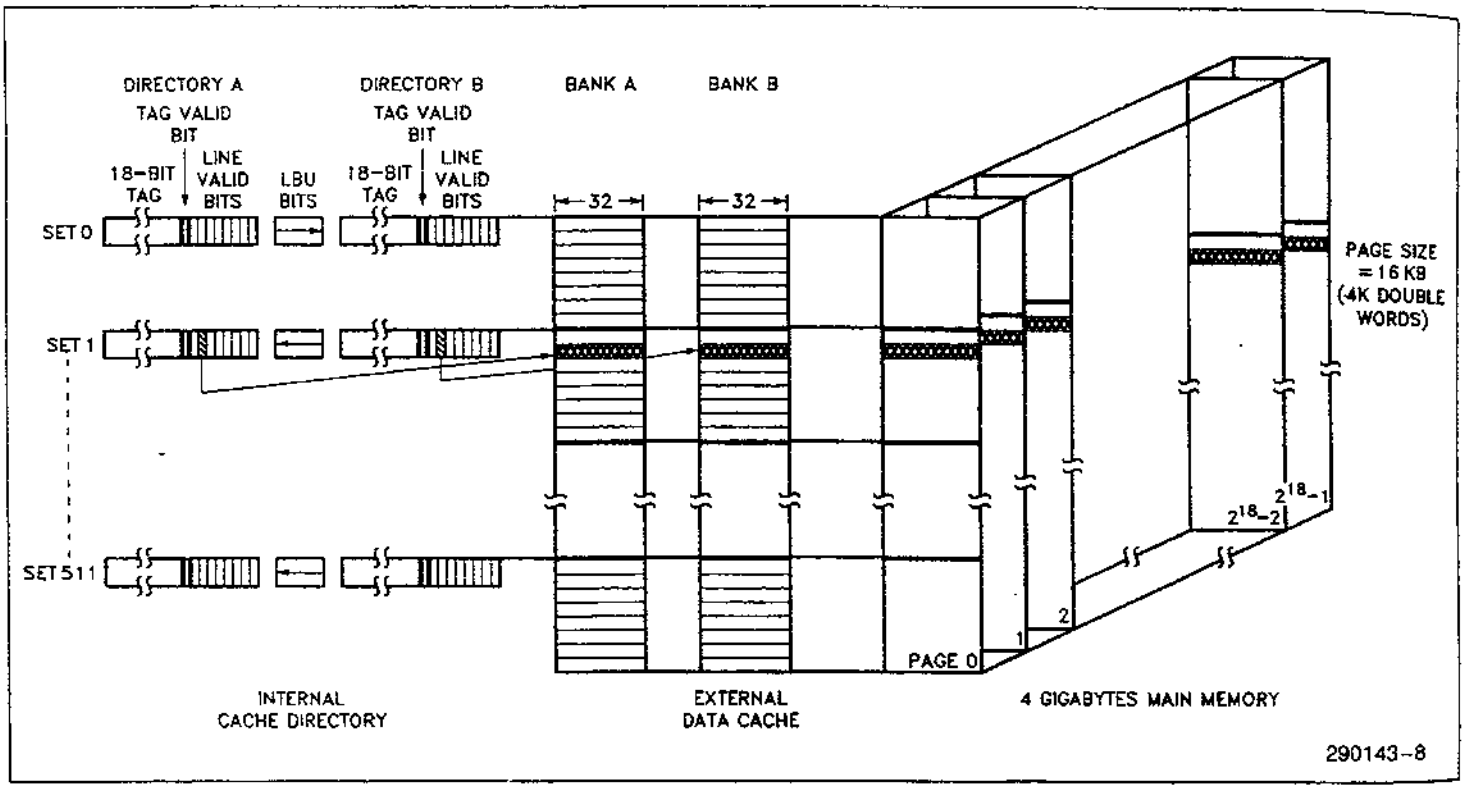
Lugemine plokkassotsiatiivsest cache-ist.

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.

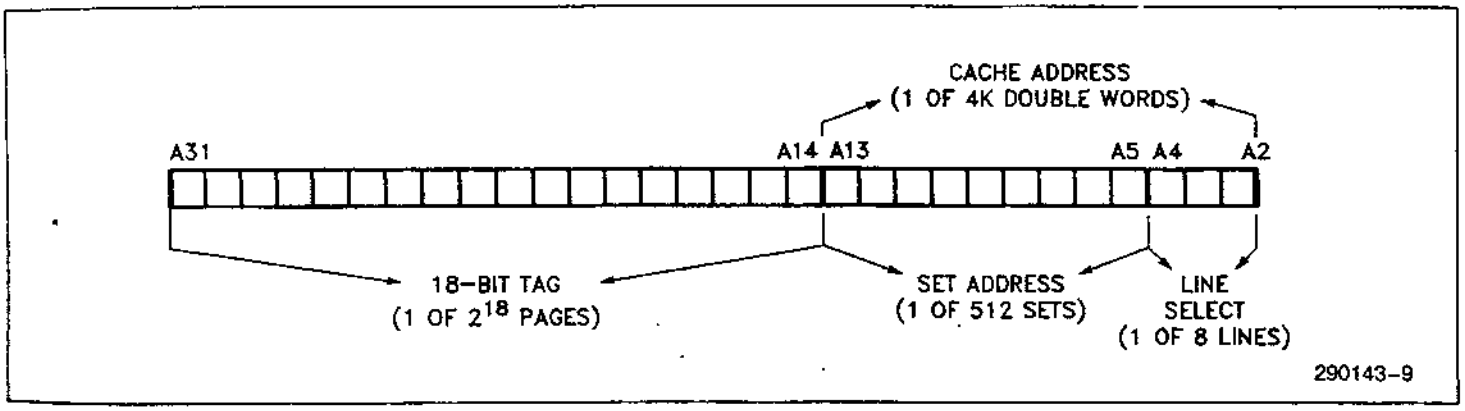
Salvestamine cache-i.

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

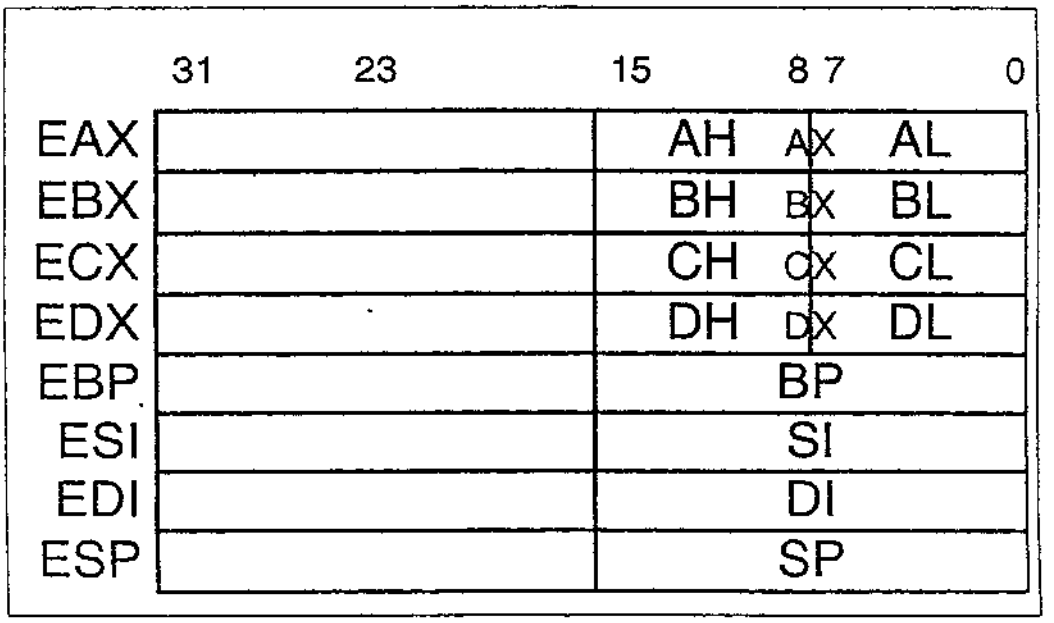
salvestus säilitab samuti koherentsuse. Tagasisalvestus rikub koherentsust, kuid ei sea selle puhul ainult cache-is olevaid andmeid kaotsimineku ohtu, kuna muudetud rea cache-ist kõrvaldamise eel salvestatakse muudetud sõna põhimällu. Oht tekib siis, kui protsessor jagab ühist põhimälu mõne teise aktiivse 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

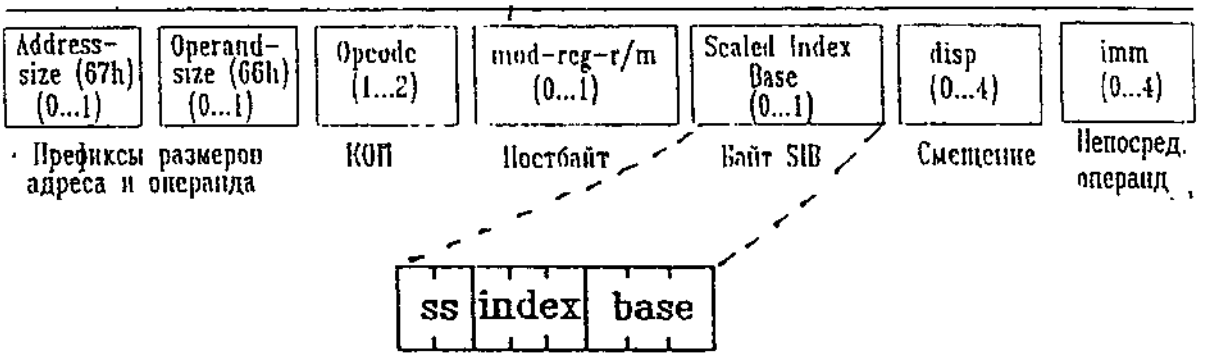
CISC –tüüpi protsessorite musternäiteks on firma Intel toodang alates mudelist 8086, mis valmis 1978.a. ja lõpetades Pentium 4-ga. Kuna Inteli protsessorid on olnud ja on ka praegu personaalarvutites enim kasutatavad, siis on loogiline käsitleda CISC protsessorite arenguga seotud probleeme just nende 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

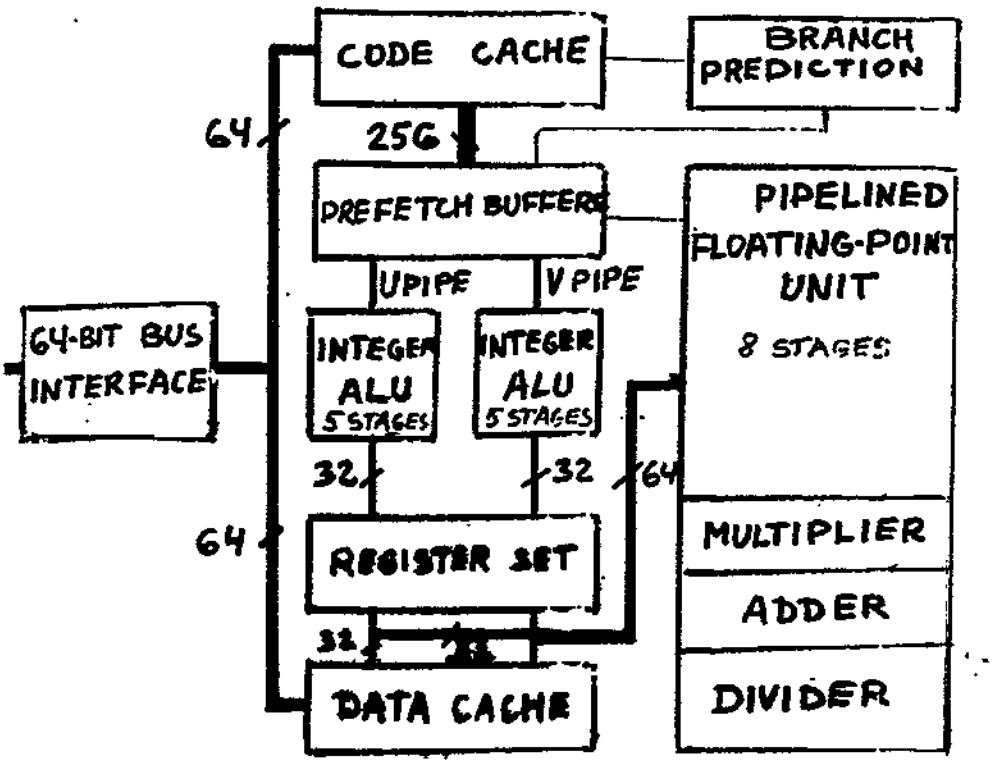


Joonis 4.1



Joonis 4.2

PENTIUM 1993

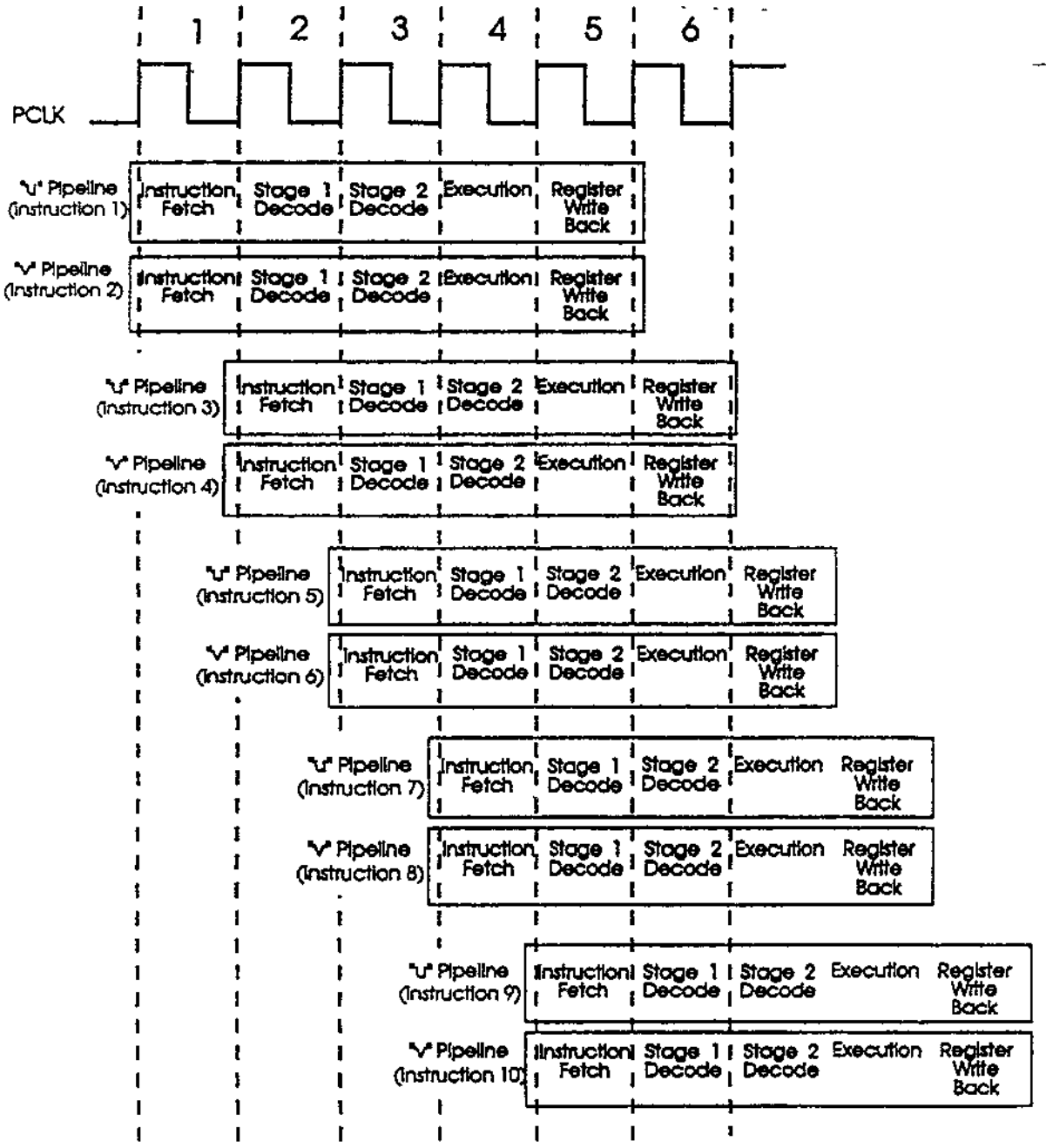


U ja V konveier	PF	D1	D2	E	WB			
Ujuvkoma konveier	PF	D1	D2	OF	X1	X2	WF	ER

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.

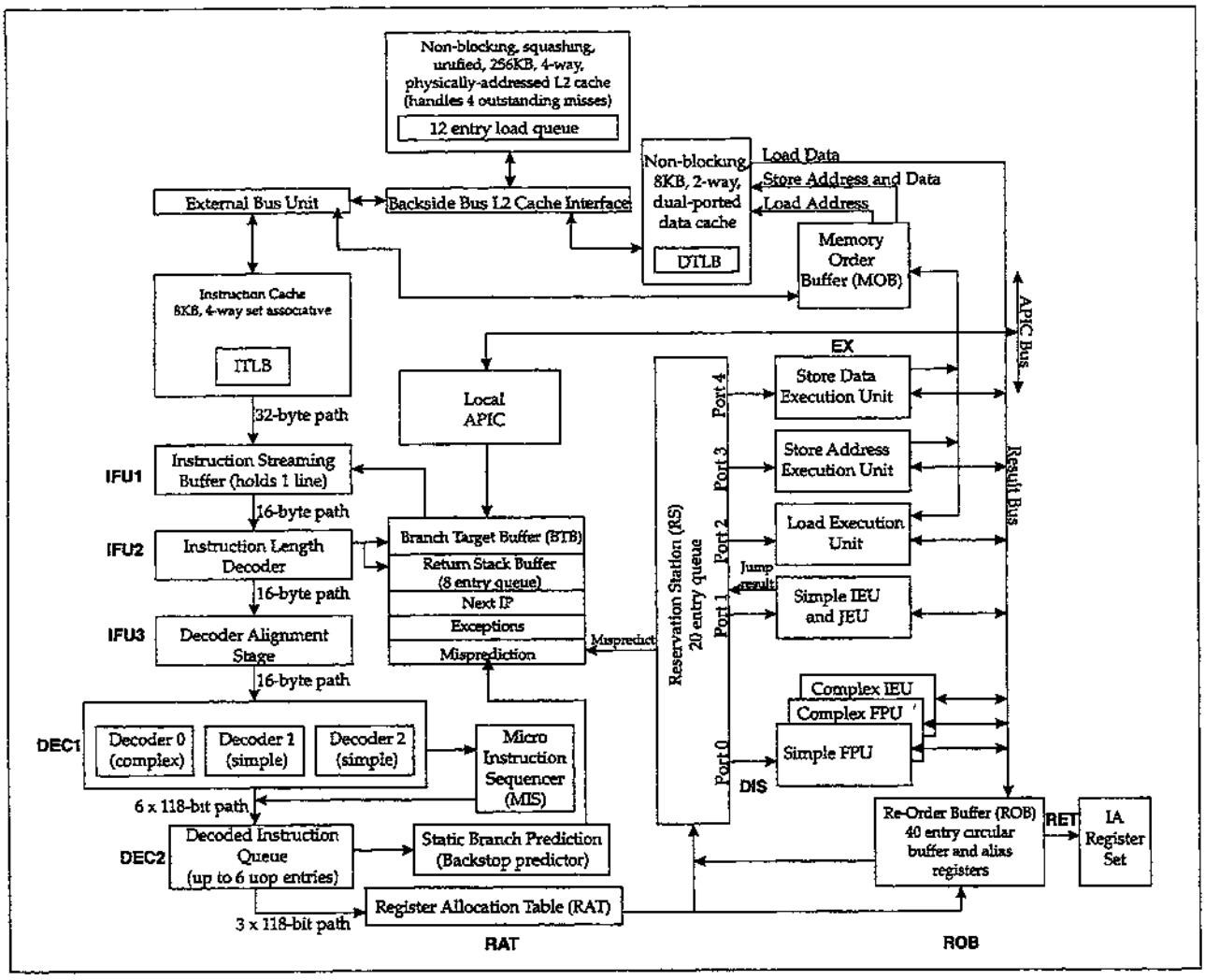
Käskude töötlemine P6 mikroarhitektuuri 11-astmelisel konveieril (joonis5.4) toimub astmete kaupa nii nagu alljärgnevalt kirjeldatud.

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.

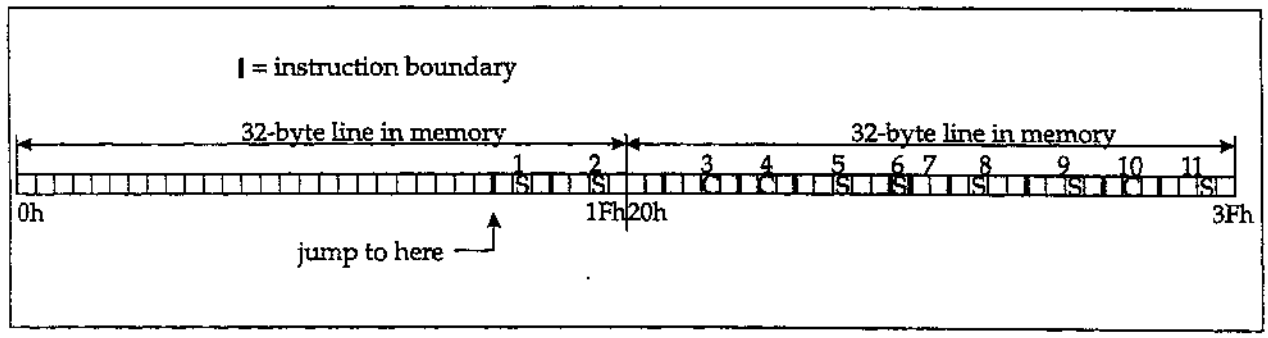
IUF2: Identifitseeritakse ja märgistatakse käskude piirid esimeses 16-baidises plokis. Kontrollitakse, kas plokis on hargnemiskäskusid. Kui on, siis saadetakse nende käskude aadressid hargnemisennustuse ploki puhvrisse (Branche Target Buffer) hargnemise ennustamiseks.

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.



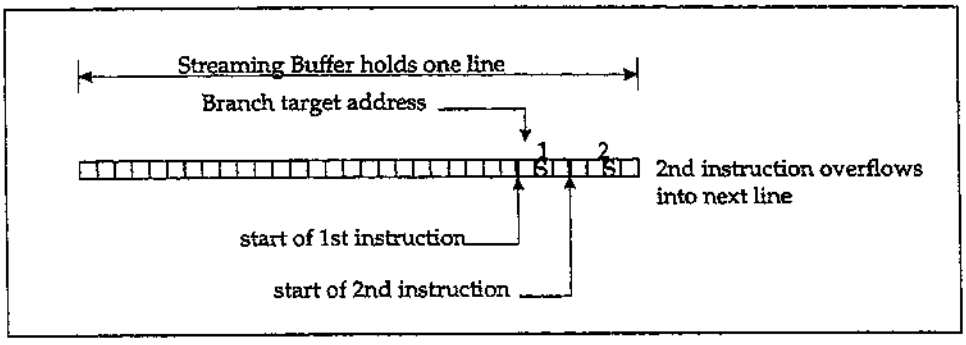
Joonis 5 .1

Example Instructions in Memory



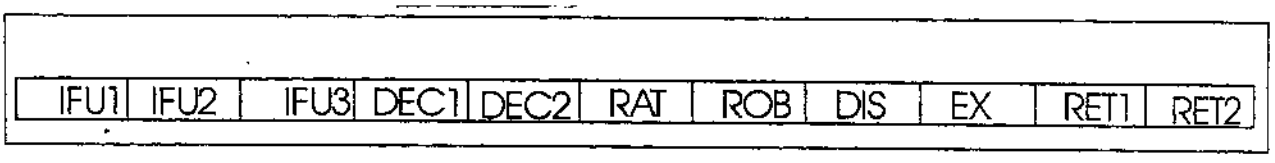
Joonis 5.2

Contents of Prefetch Streaming Buffer Immediately after Line Fetched



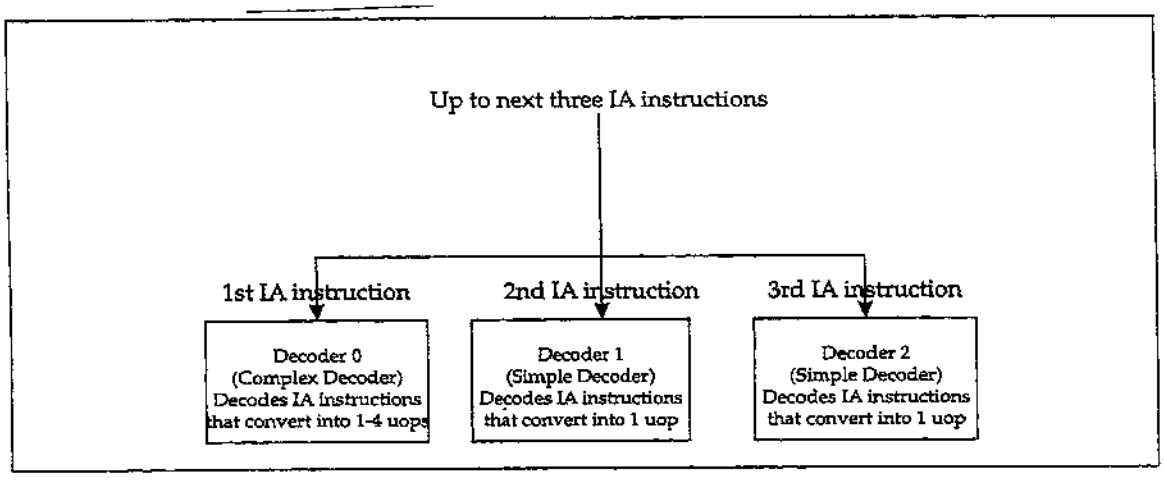
Joonis 5.3

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:

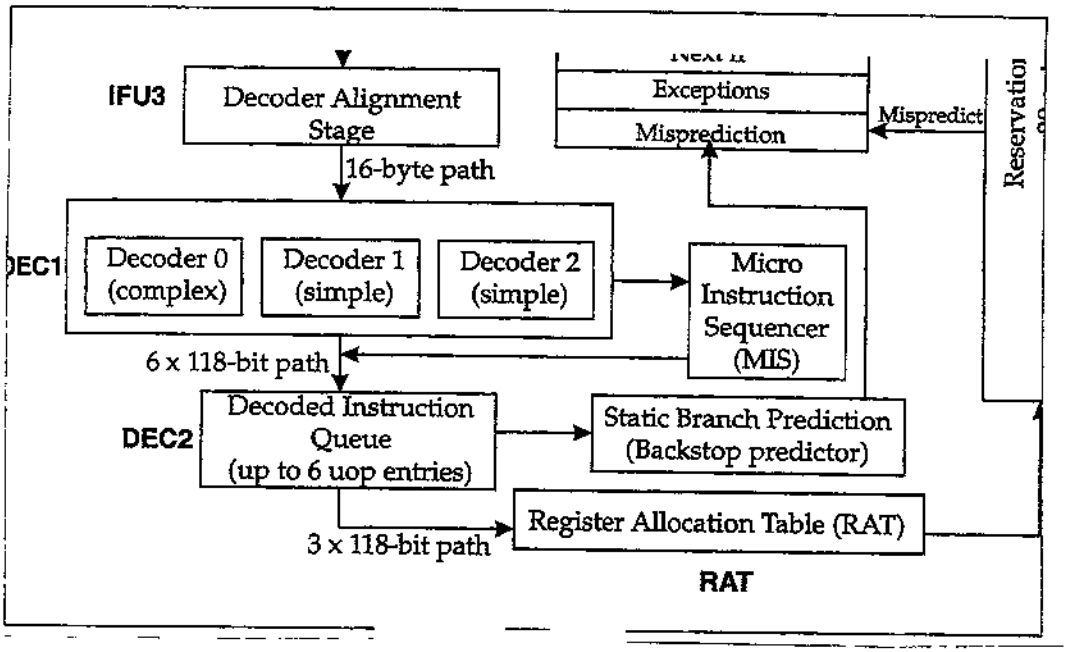
- register – register tüüpi käsud;
- mälust lugemise käsud;

Kaheks mikro-opsiks teisendatakse :

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

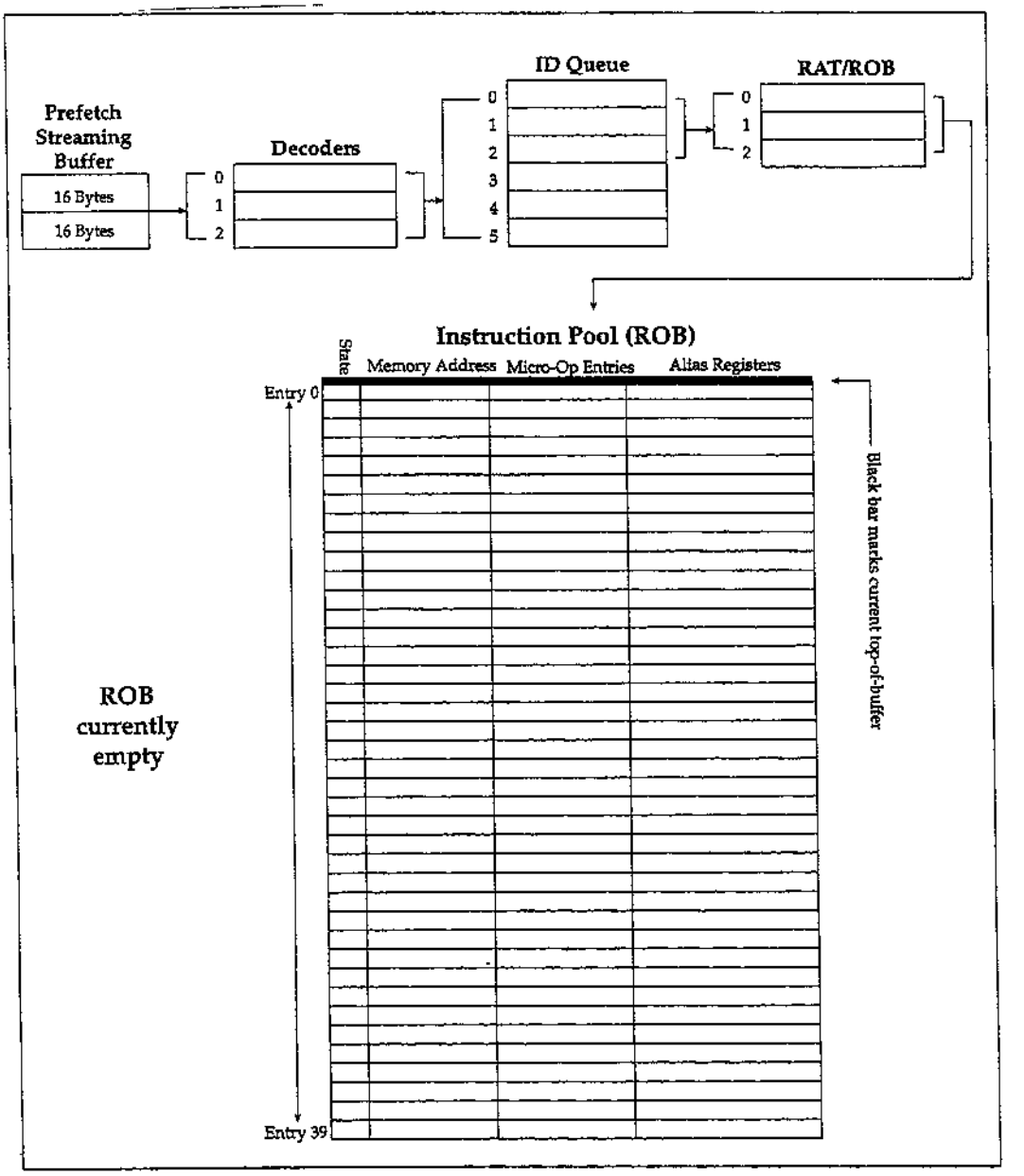
Kaheks või kolmeks mikro-opsiks teisendatakse register/ modifitseerimise/ mällu salvestamise käsud.
Neljaks mikro-opsiks teisendatakse mälust lugemise/ modifitseerimise /mällu salvestamise käsud;
Kasutades IFU2-s paika pandud käskude piirimärgendeid, reastatakse käsud IFU3-s dekooderitega vastavalt nende 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

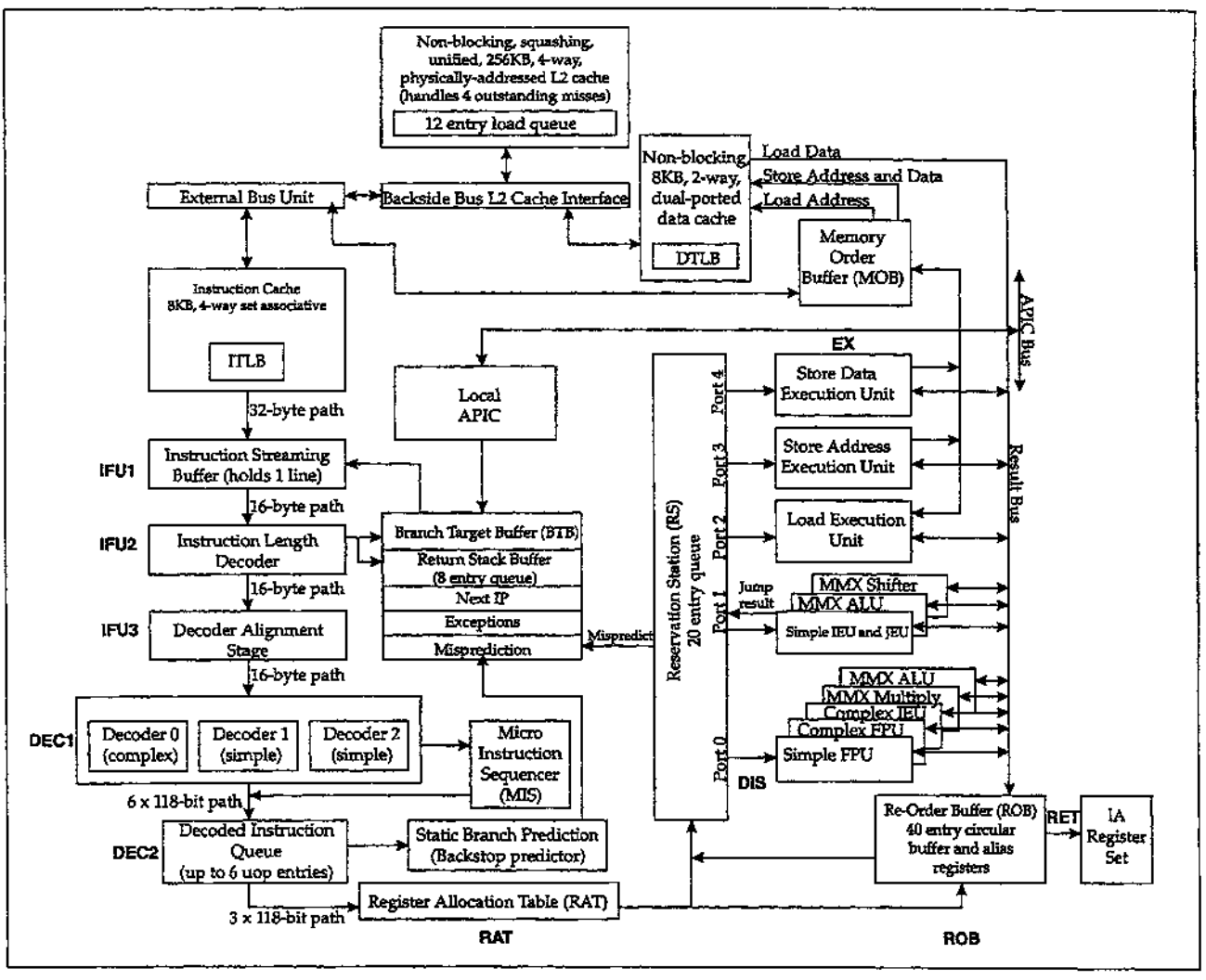
DISP ja **EX:** Varustusjaam (Reservation Station – RS) kopeerib üheaegselt kuni viis mikro-opsi ja paneb need järjekorda kuni nende sisestamiseni vastavatesse sooritusüksustesse. Mikro-opside varustusjaamast sooritusüksustesse väljastamise kriteeriumiks on täitmiseks vajalike lähteandmete ja vastavate 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.

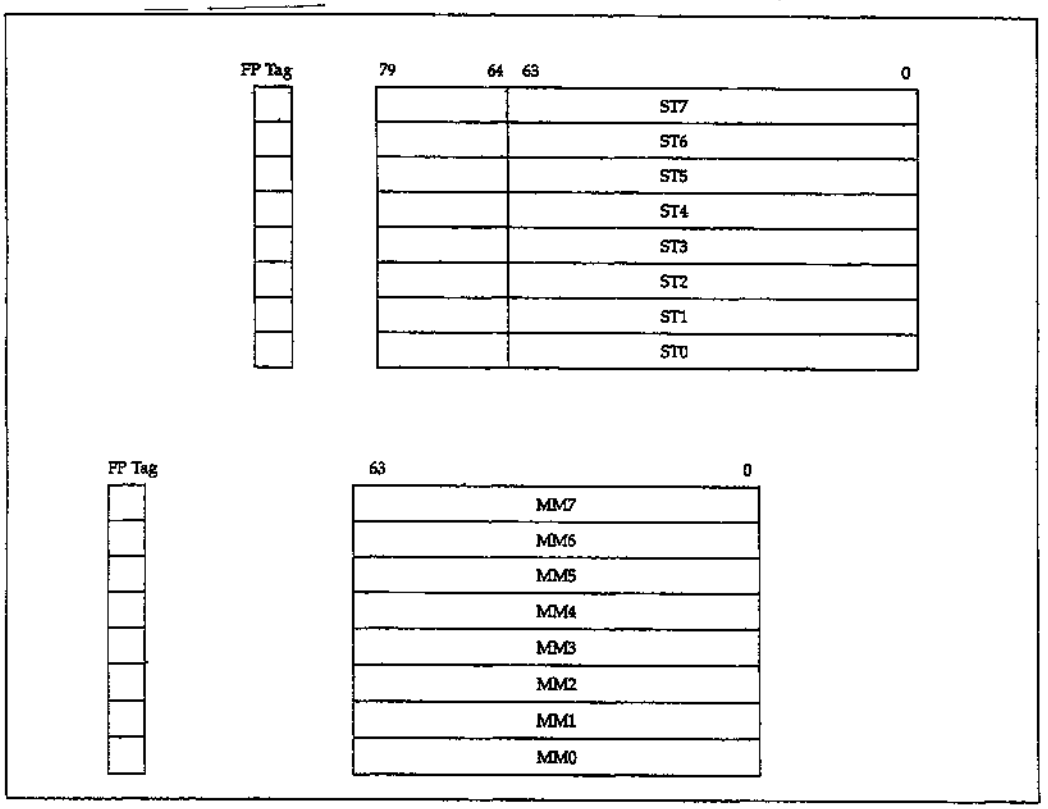
5.3 PENTIUM III

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.

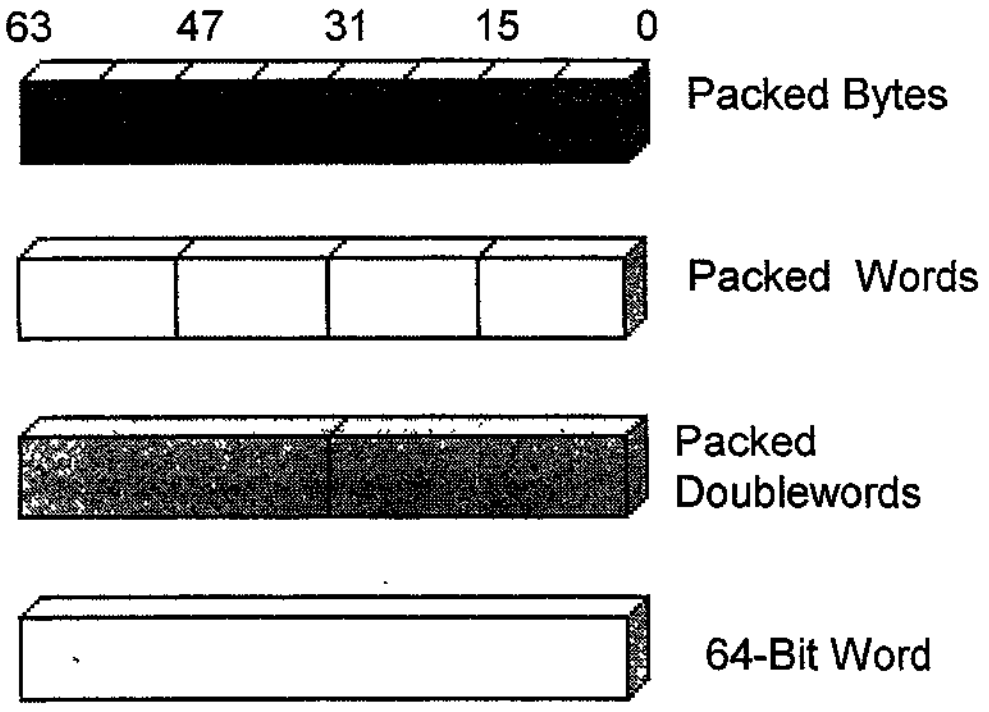

Non-blocking, squashing, unified, 256KB, 4-way, physically-addressed L2 cache (handles 4 outstanding misses)
12 entry load queue
Non-blocking, 8KB, 2-way, dual-ported data cache
DTLB
Load Data
Store Address and Data
Load Address
Memory Order Buffer (MOB)
External Bus Unit
Backside Bus L2 Cache Interface
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

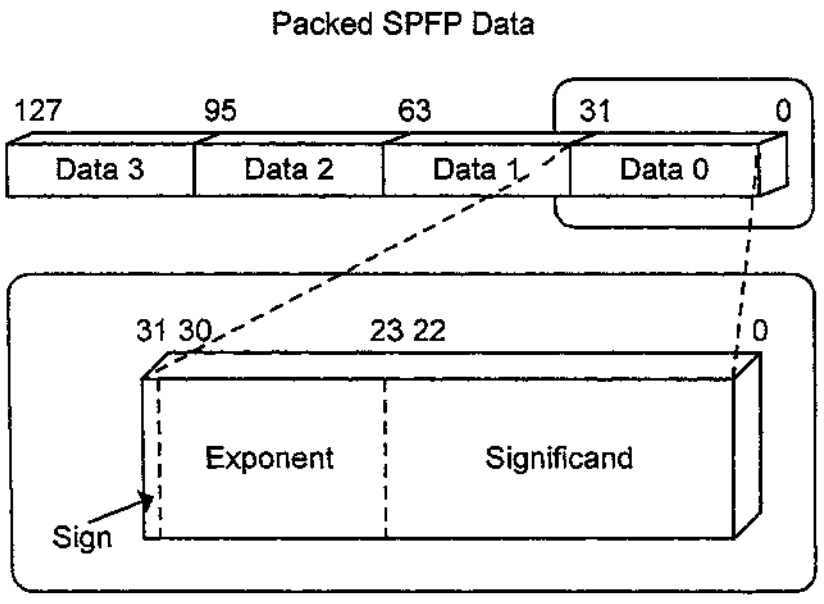


Joonis 5.10



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;

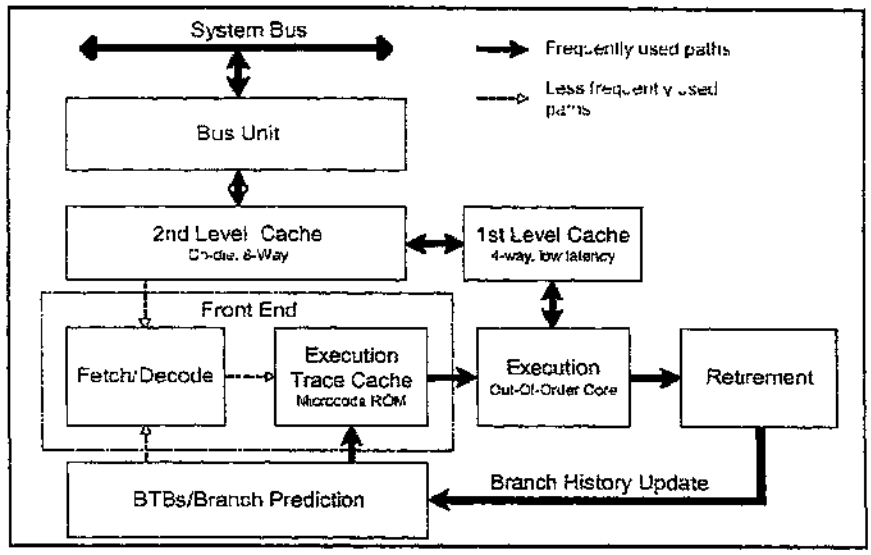
Pentium 4 lihtsustatud plokkskeem on kujutatud joonisel 5.12 , milles on kasutatud järgmisi uuendusi:

- 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);

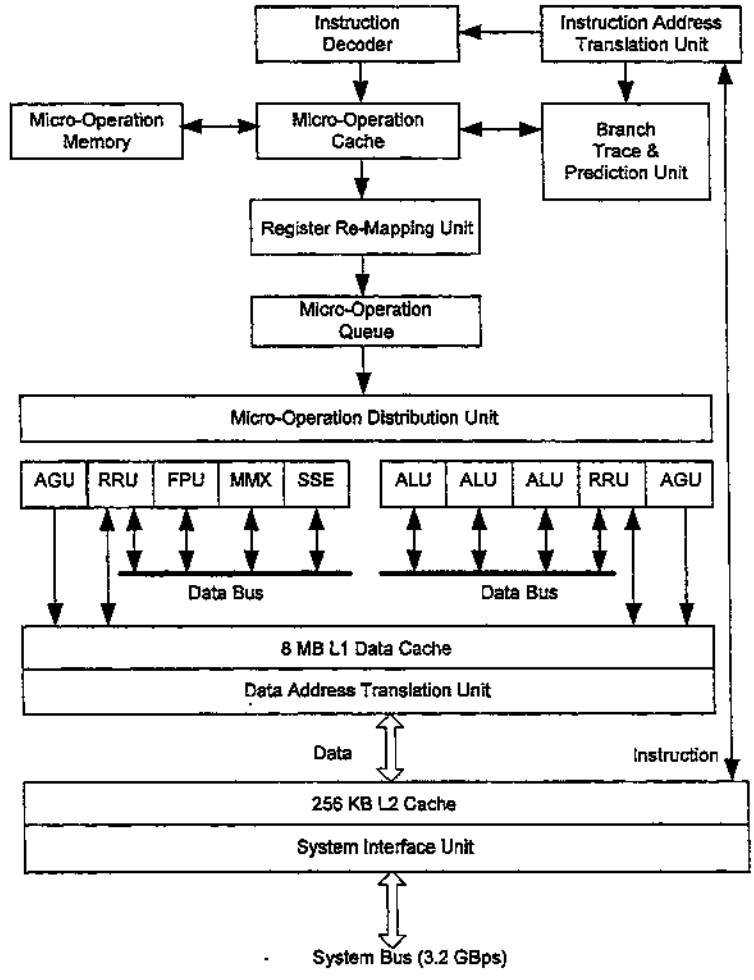
Pentium 4 struktuurskeem on toodud joonisel 5.13. L2 cache on plokk-assotsiatiivne 8-suunaline tagasisalvestusega cache . Käskude dekooder koos mikroprogrammi mäluga muudab IA käsud mikrokäskude jadadeks, mis seejärel salvestatakse trace cachei, mille maht on 12000 mikrokäsku. Mikrokäsud on cacheis järjestatud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

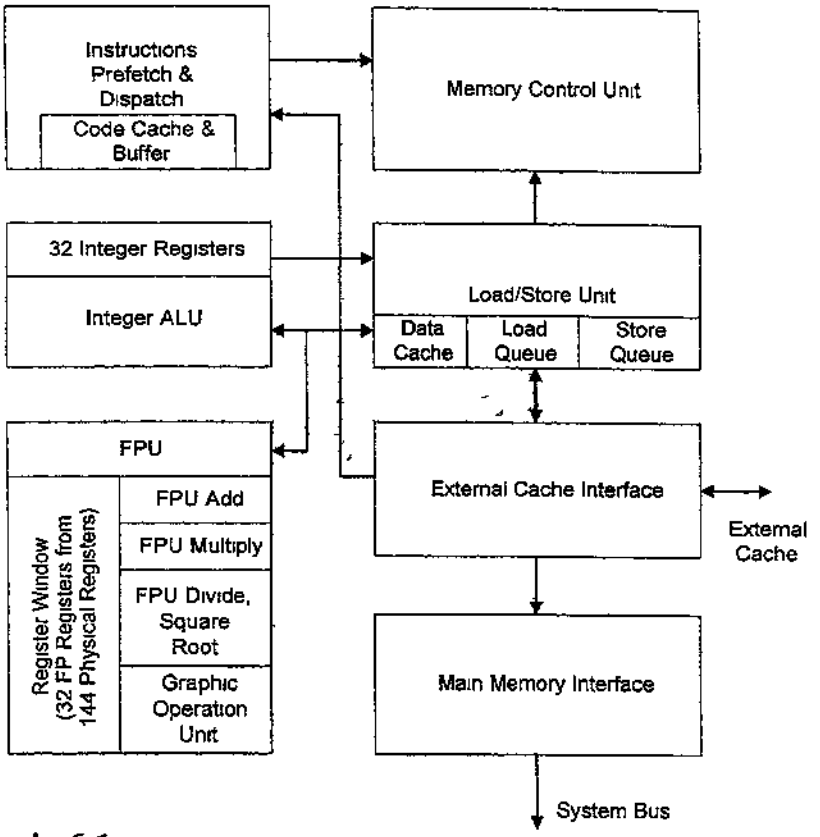
UltraSPARC-i käskude konveier on üheksaastmeline ja võimaldab täita kuni neli käsku igal taktil. Protsessor ei muuda käskude täitmise järjekorda. Igal taktil võib protsessor täita kaks tehet täisarvudega, kaks ujukomaarvudega, ühe laadimis/salvestuskäsu ja ühe hargnemiskäsu. Olgugi et kokku on võimalik 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.

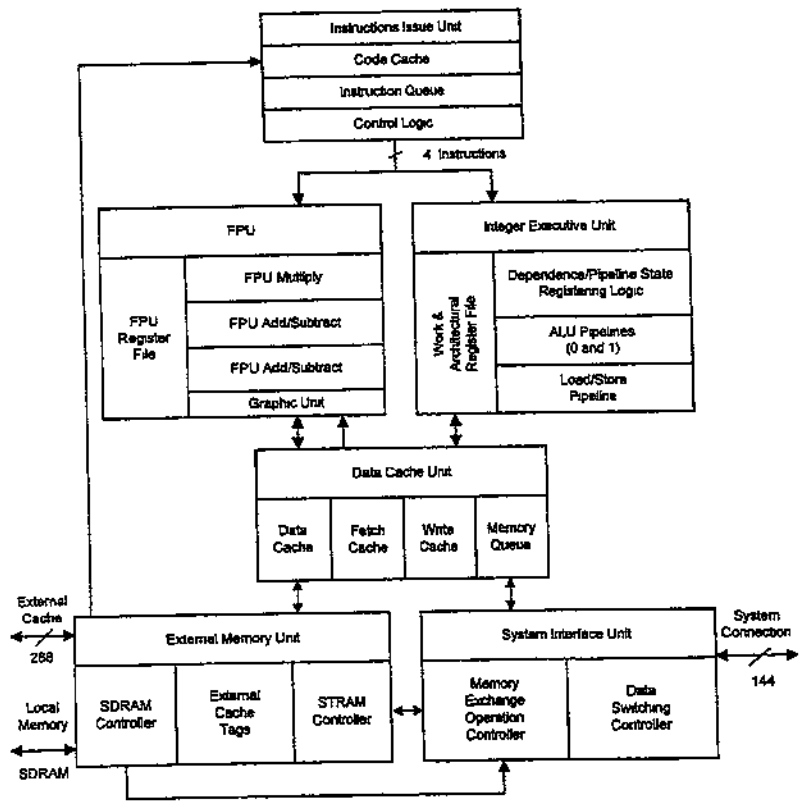
Sarnaselt eelmiste põlvkondadega on UltraSPARC III-s rakendatud haru ennustust, käskude tasemel parallelismi (Instruction- level Parallelism-ILP) avastamist kompileerimise tasemel ja registeraknaid (Register Windows). Käskude täitmise konveier on 14-astmeline.

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

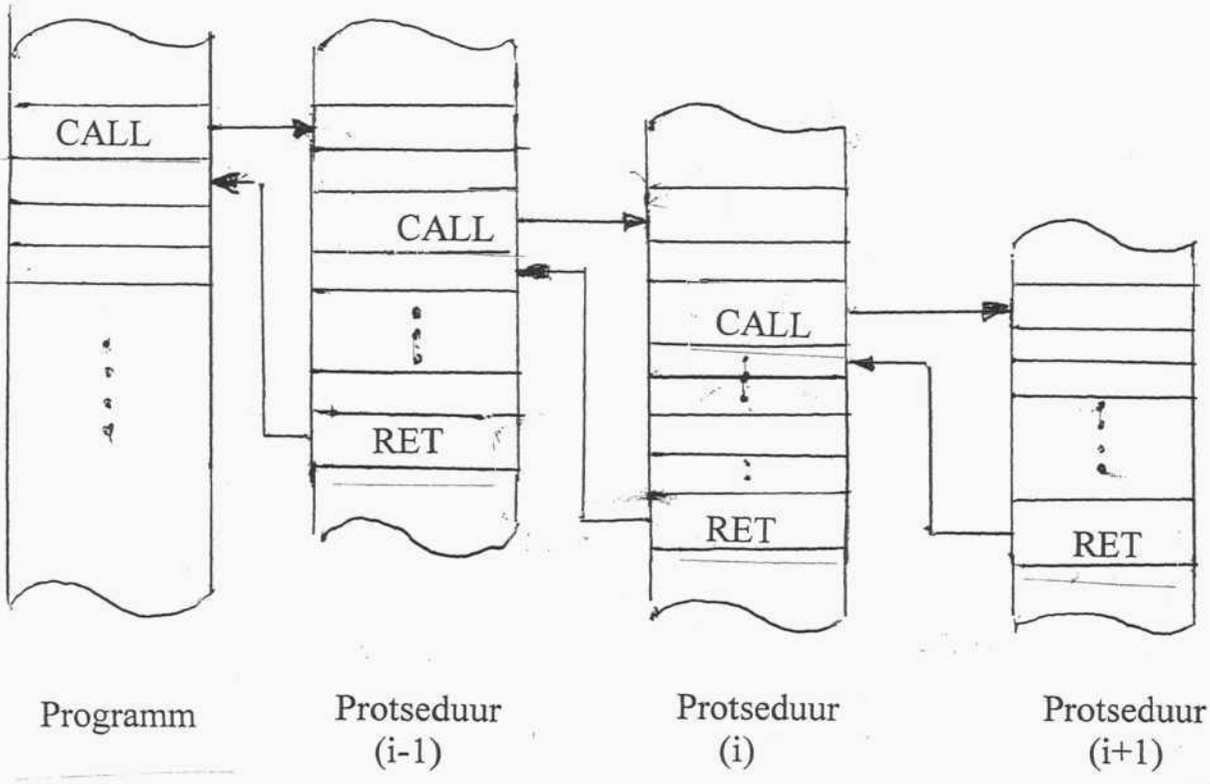


Joonis 6.2 UltraSPARC III microprocessor architecture.

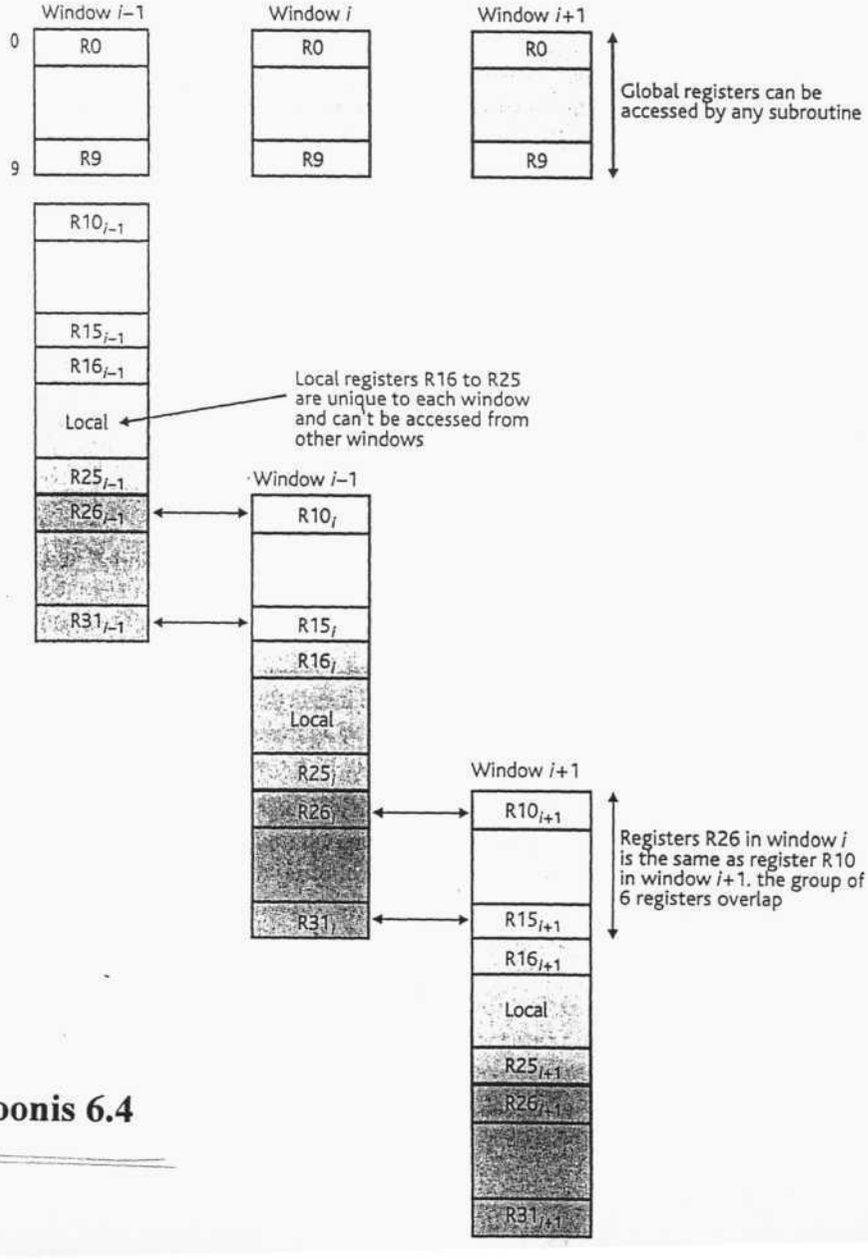
Käsuvõtuplokk (Instruction Issue Unit – IIU) ennustab hargnevusi ja valib täitmiseks käskusid, võttes arvesse ennustatud haru. Valitud käsud paigutatakse järjekorda kahte funktsionaalplokki: IEU-sse ja FPU-sse.
IIU on varustatud 4-suunalise plokk-assotsiatiivse 32 KB-se käskude cache-iga, TLB-ga (Translation 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.

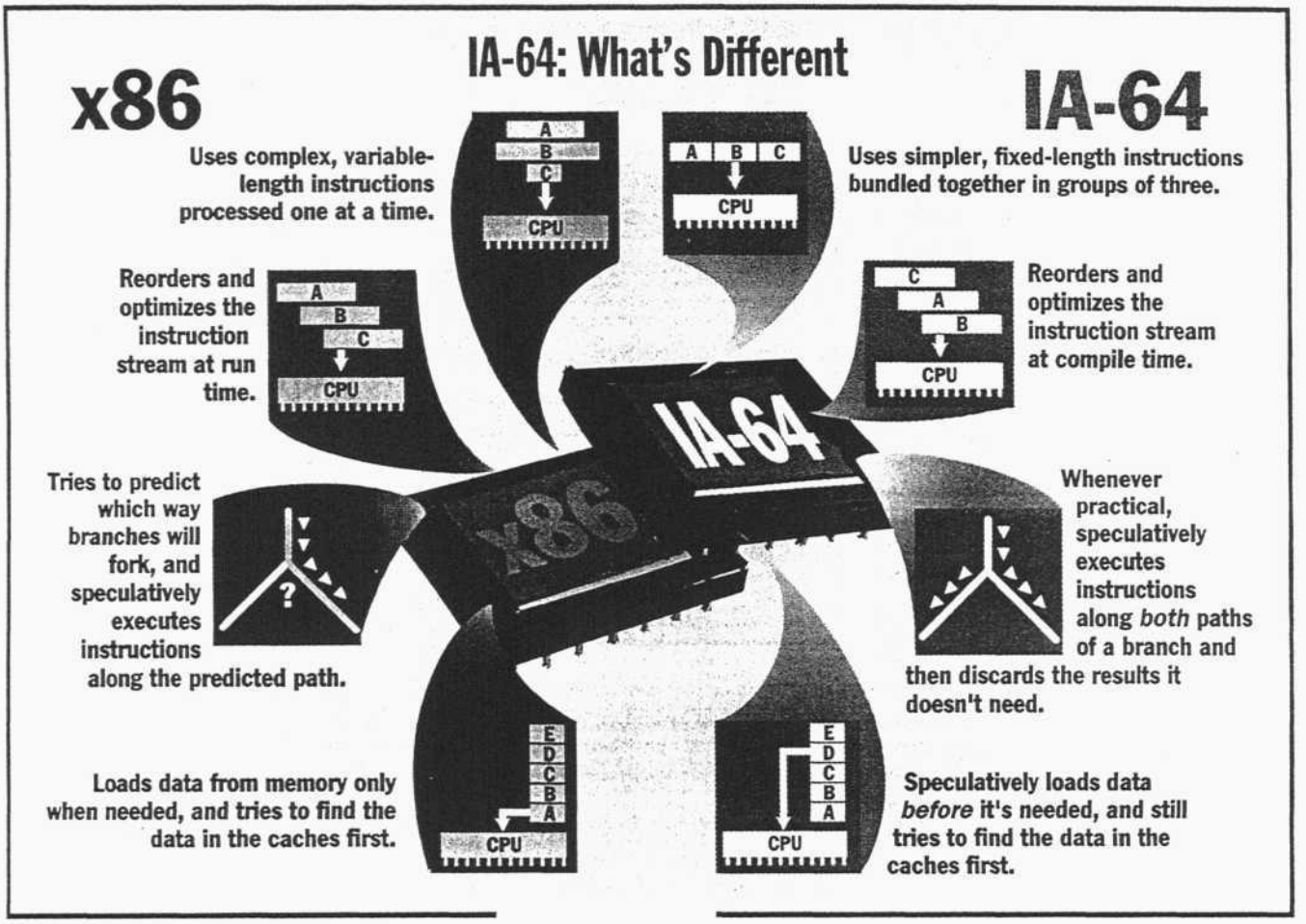
7.1 Väga pika käsusõnaga protsessori arhitektuur

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.

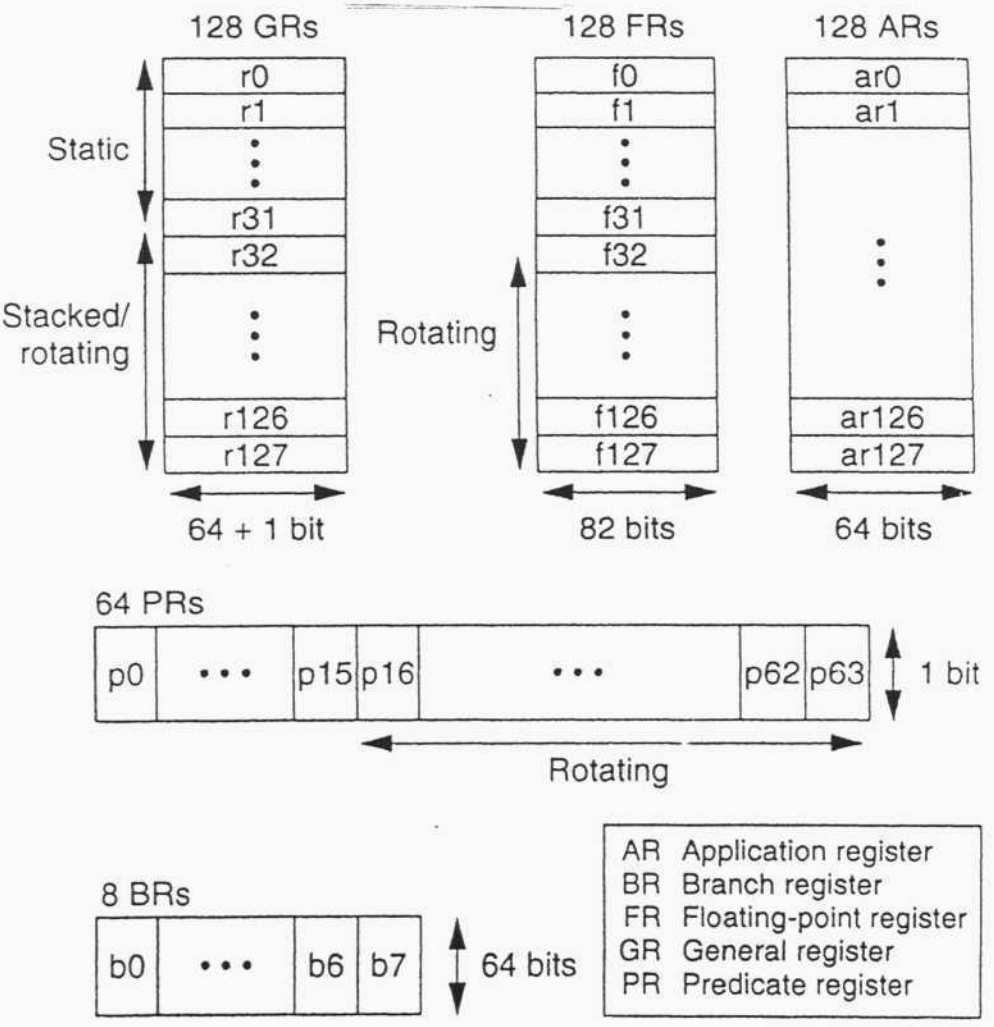
IA-64 käsustik koosneb 128-bitistest käskude pundardest (bundles), millest igaühes kolm 41-bitist RISC-tüüpi käsku. Iga pundar kätkeb endas informatsiooni käskude omavahelise sõltuvuse kohta, mis on kindlaks tehtud programmi kompileerimisel. Seda informatsiooni kasutab protsessori riistvara selleks, 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



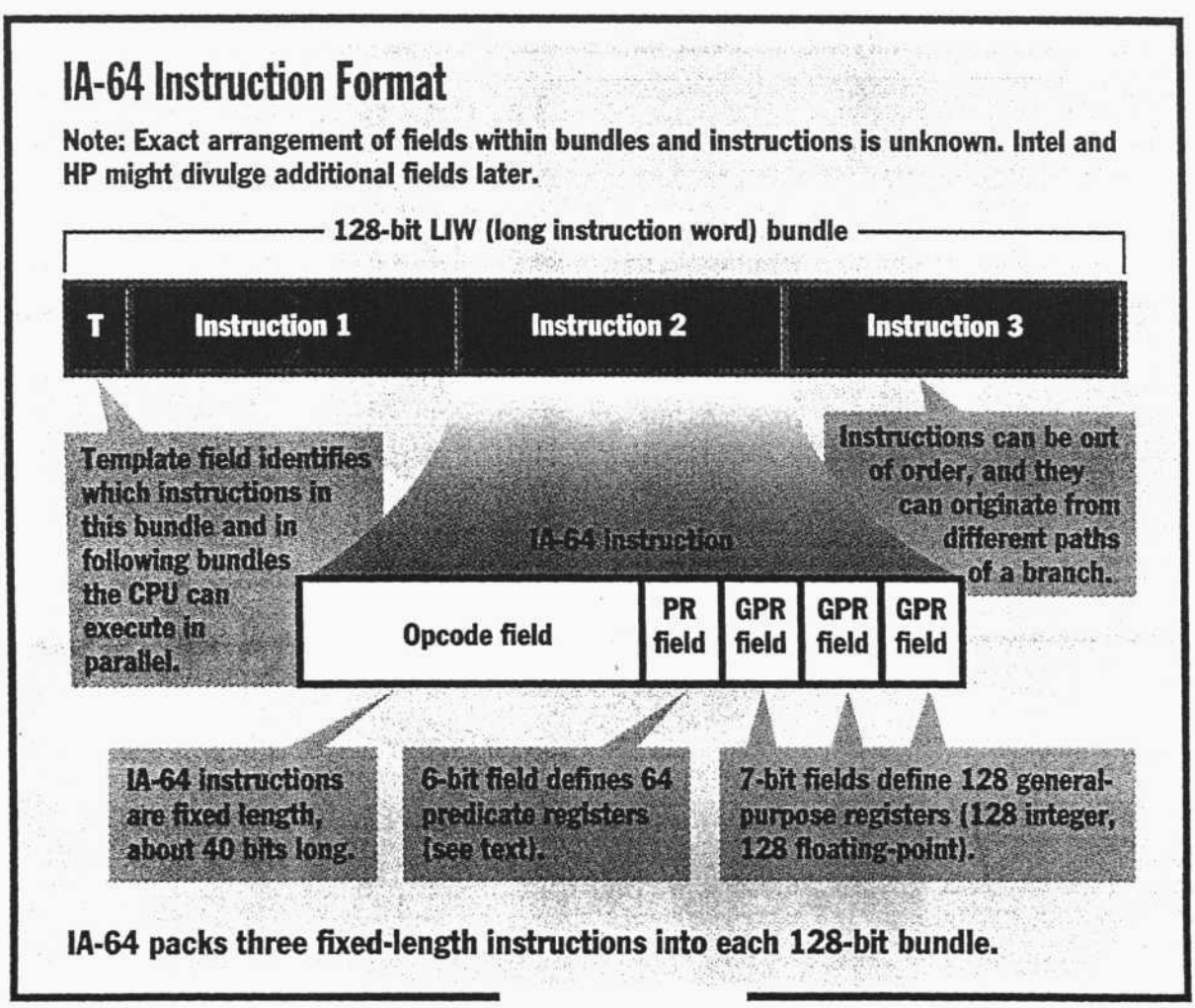
IA-64 application state. **Joonis 7.2**

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:

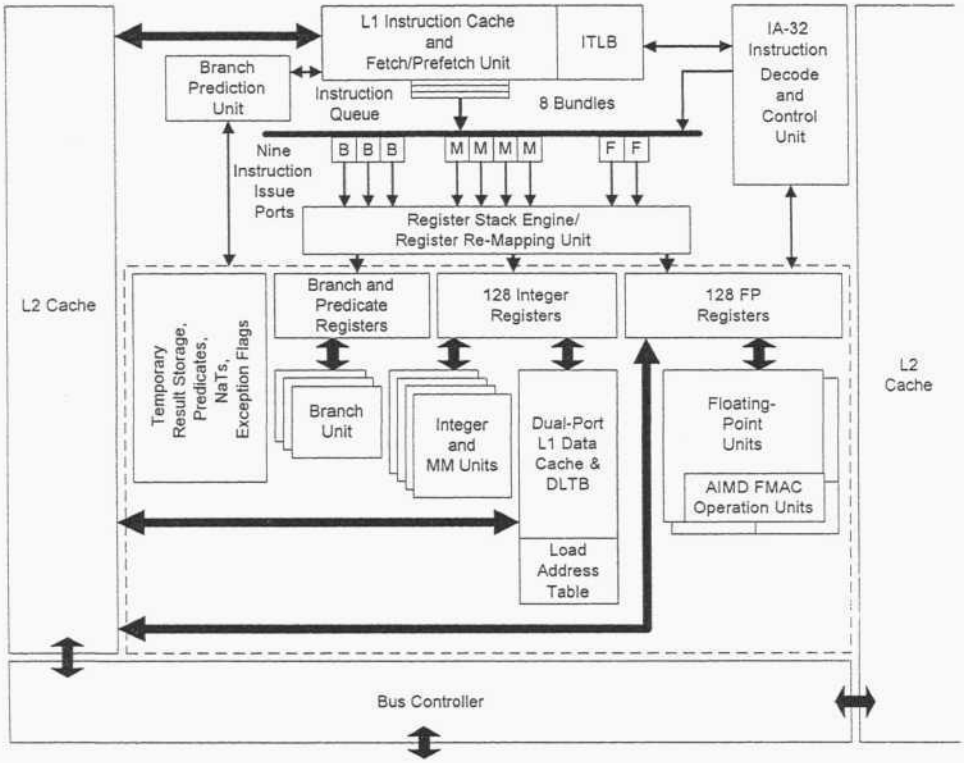
- 4 täisarvulise aritmeetika plokki;
- 4 multimeedia andmete töötlusplokki;
- 2 ühekordse täpsusega ja 2 laiendatud täpsusega ujukoma plokki;
- 2 laadimise /säilitamise plokki;
- 3 hargnemise plokki;

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;

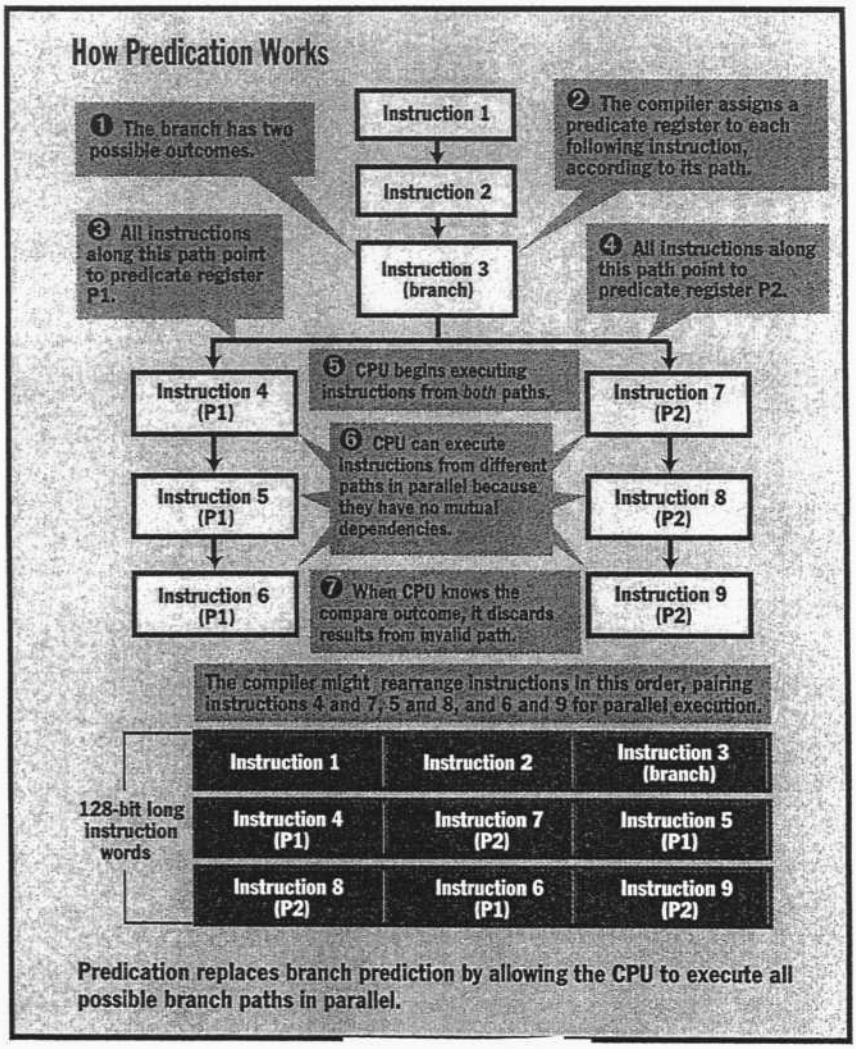


Joonis 7.3

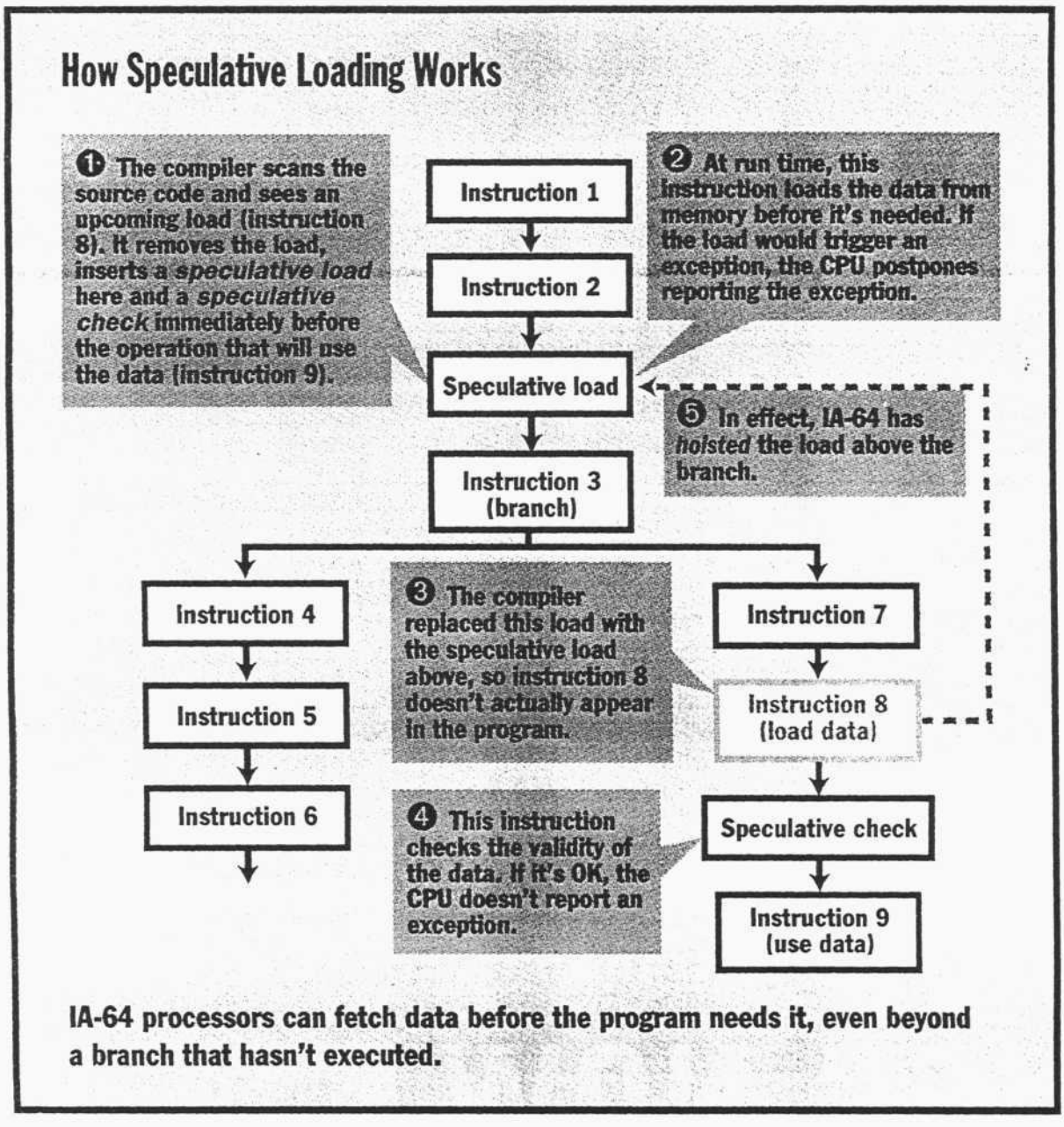


Itanium microprocessor structure block diagram.

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.

7. 2 **Mitmelõngalise protsessori arhitektuur** (Multithread Processor Architecture)

Arvutitehnilises kontekstis pole kerge leida ingliskeelsele terminile "multithread" sobivat eestikeelset vastet. Sõna ***thread*** tähendab tõlkes niiti, lõnga või seost. Eesti keeles on kasutusel selline mõiste nagu ***mõttelõng,*** mis väljendab inimajus tekkivat mõtete jada. Arvutiprogrammi täitmist protsessoril võib vaadelda, kui mõtlemise analoogiat. Seepärast on salvestatud programmiga arvutite leiutamisest alates neid populaarteaduslikus kirjanduses nimetatud ka "mõtlevateks masinateks" . Seega peaks termin ***mitmelõngaline e. multilõngaline*** arvutitehnilisse konteksti sobima küll. Näib, et erinevalt mitmelõngalisest protsessorist võib inimene mistahes ajavahemikul arendada siiski vaid üht mõttelõnga korraga.

Sisuliselt võimaldab multithread (mitmelõngaline ehk multilõngaline) protsessor, võrreldes single-thread (ühelõngalise) protsessoriga, täita paralleelselt (s.o. üheaegselt) mitut erinevat programmi või ühe programmi mitut üksteisele järgnevat lõiku, kasutades protsessori sooritusplokkide (ALU-d, FPU-d, MMX-, XXM-, salvestus- ja laadimisplokid) jõudeolekuid, mis paratamatult tekivad programmi täitmisel ühelõngalisel superskalaarsel või väga pika käsusõnaga (VLIW) protsessoril, ükskõik kui hästi seejuures programm ka ei oleks käskude paralleeltäitmisele orienteerituse mõttes kompileeritud. Seega erinevate tehete sooritamiseks ettenähtud plokkide jõudeolekute arvel saab paralleelselt täita kahte või enamat programmi. Paralleelsus saavutatakse erinevatest programmidest pärit tehete üheaegse sooritamisega. Selleks aga peab olema tagatud neist programmidest üheaegne sõltumatu käsuvõtt.

Selleks, et mitme erineva programmi üheaegne täitmine oleks sõltumatu, peab protsessoris olema lõngade arvuga võrdne arv programmiloendureid ja igat lõnga teenindav omaette registrifail . Näiteks kahelõngalises protsessoris on kaks programmiloendurit ja kaks adresseeritavat registrite faili.

Protsessori multilõngalisus võib olla teostatud mitmel erineval viisil. Näiteks, n-lõngalises protsessoris, milles on n programmiloendurit ja n registrifaili (joonis 1), saab erinevatest lõngadest pöörduda ühisesse mällu (mis on aeglane protsess) kindlas järjekorras võrdsete perioodide järel, alustades esimesest ja lõpetades n-daga ning alustades seejärel uuesti esimesest jne., kusjuures mälu poole pöördumise tsükli kestus peab mahtuma n –1 perioodi sisse. Seega tuleb lõngade arv n valida selline, et mällu pöördumise aeg oleks lühem kui n – 1 protsessori taktiperioodi. Sel ajal kui antud lõng toimetab mäluga, aeglustub selle töö n korda. Samal ajal saavad ülejäänud n –1 lõnga üheaegselt toimetada igaüks oma registrites olevate andmetega.

Lisaks käsuloendurile ja registrifailile peab protsessoris olema iga lõnga jaoks oma käsuvõtu plokk, mis määrab käskude täitmise akna ning vahendid hargnevuste ennustamiseks, registrite ümbernimetamiseks, mis on vajalikud

käskude dünaamiliseks mittejärjekorras täitmiseks ja tulemuste programmiga määraud järjekorras väljastamiseks

Lõngad identifitseeritakse programmi kõrgtasemelise lähtekoodi või objektkoodi analüüsi käigus enne nende täitmist. Kuid seejuures ei suuda kompailer alati lahendada lõngade omavahelisi sõltuvusprobleeme , mis tekivad registrite ja mälupesade ühiskasutusest. Neid sõltuvusi tuleb lahendada lõngade täitmise käigus. Sel eesmärgil varustatakse protsessor lõnga spetsiaalse tingimusliku täitmise aparatuuriga, mis võimaldab lõngade sõltuvuse avastamisel pöörata täitmine tagasi ja tühistada tulemused. Sõltuvusvea näitena võib tuua olukorra, kus üks lõng üritab salvestada mälupesasse, millisest teine lõng parajasti sooritab lugemisoperatsiooni, mis oleks pidanud toimuma järgmisel tsüklil. Sellisel juhul viga registreeritakse ja teine lõng suunatakse punkti, kust saab lugeda korrektse väärtuse.

Multilõngalise protsessori aparatuuri igat lõnga eraldi töötleva osa ja kõiki lõngasid töötleva ühisosa (s.o. tehteid sooritavate plokkide) vaheline liides võib olla realiseeritud aparatuuri erineval tasemel. See võib olla paigutatud kas käsuvõtuplokkide väljunditele (s.o. sooritusplokkide sisenditele), või siis mällu pöördumise tasemele. Sooritusplokkidega vahetult seotud liides võimaldab efektiivselt täita tugeva lõngadevahelise sõltuvusega järjestikuseid programme. Sellise liidesega protsessor kannab nimetust **simultaanne multilõngaline protsessor** (simultaneous multithread processor – SMT).

Mällu pöördumise tasemel realiseeritud liidesega protsessor on orienteeritud täitma järjestikustes programmides sõltumatuid või nõrgalt seotud lõngasid. Selline protsessor sarnaneb rohkem mitut programmi paralleelselt täitva ühekiibilise multiprotsessorsüsteemiga.

Firma Intel võttis 2002. aastal simultaanse multilõngumise tehnoloogia kasutusele oma Xeon protsessorite perekonnas nimetades seda ***hüperlõngumise tehnooogiaks*** (Hyper – Threading Technology). Intel Xeon protsessori arhitektuur on kahelõngaline, mis tähendab, et üks füüsiline protsessor on vaadeldav kahe loogilise protsessorina, mis täidavad üheaegselt käskusid kahest käskude jadast (lõngast) kasutades selleks ühiseid sooritusplokke (ALU-d, FPU-d, MMX-d jne.). Tarkvara seisukohalt vaadatuna pole vahet, kas need kaks lõnga täidetakse kahel loogilisel või füüsilisel protsessoril. Mikroarhitektuuri seisukohalt vaadatuna aga täidetakse lõngad nende ühiskasutuses olevatel sooritusplokkidel.

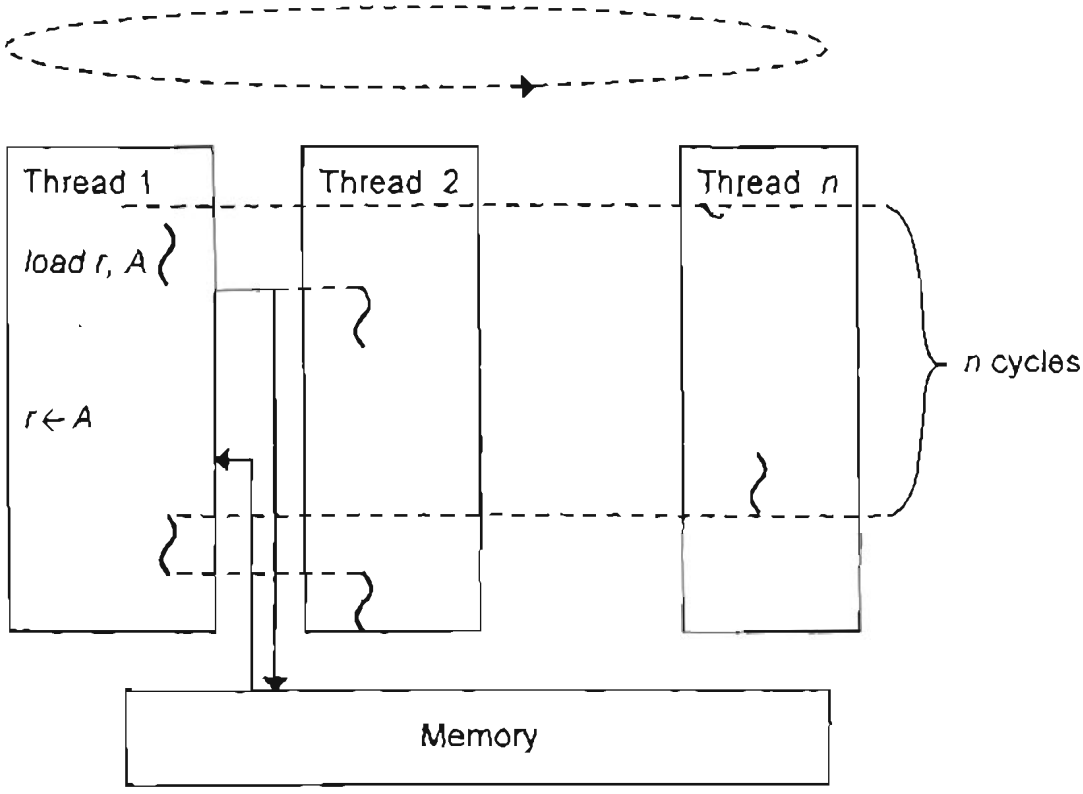
Valdava osa mikroarhitektuursetest ressurssidest on mõlema loogilise protsessori jaoks ühised ja ainult väike osa on nende jaoks realiseeritud eraldi, mis hõivab lisaks umbes 5% kristalli pinda.

Kui üks loogilistest protsessoritest mingil põhjusel seiskub, töötab teine edasi, kasutades selleks täies mahus kõiki ühiseid ressursse. Loogiline protsessor võib seiskuda erineval põhjusel, milleks võib olla cache'i

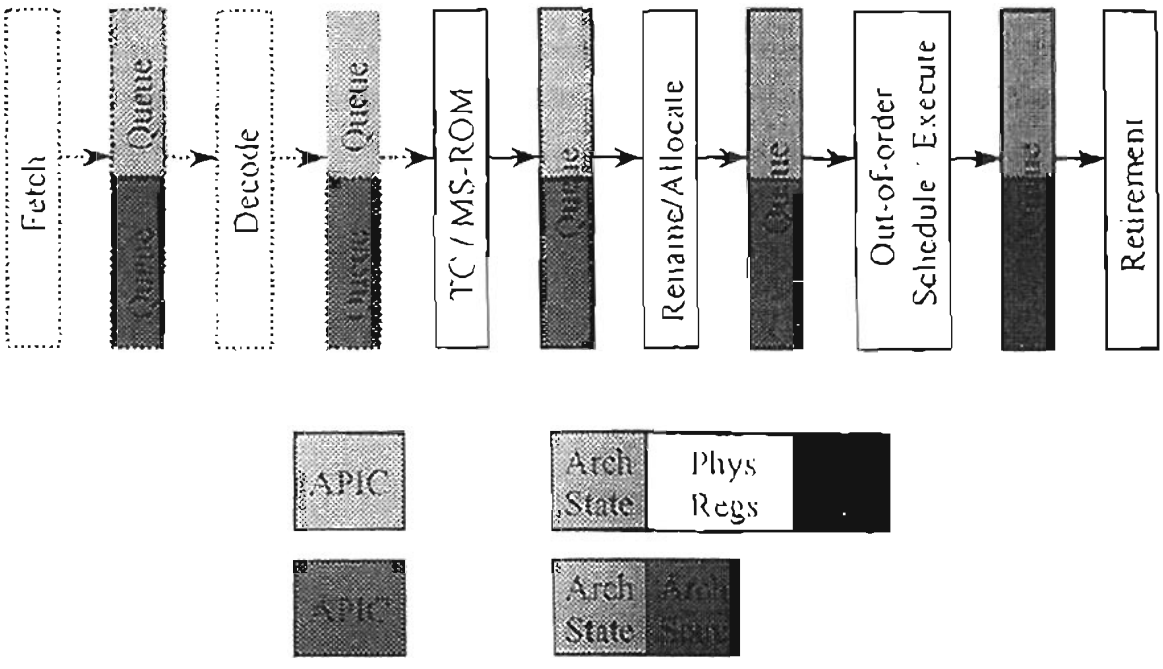
möödalask, vale suuna valik hargnemisel või eelmise käsu tulemuse ootamine.
Kui protsessor täidab ajutiselt ainult üht aktiivset lõnga, siis töötab see nagu tavaline ühelõngaline protsessor. Joonisel 2 on kujutatud konveierit mikroarhitektuuri tasemel, milles loogikaplokid on üksteisest eraldatud puhvritena toimivate järjekordadega, mis tagavad lõngade üheaegse sõltumatu edenemise konveieril. Puhverjärjekorrad on mõlemale lõngale kohati individuaalsed, kohati ühised.
Konveieri detailsema struktuuri tööpõhimõtte paremaks mõistmiseks on see jaotatud kaheks. Käsuvõtu ja dekodeerimise osa on kujutatud joonistel 3 a ja b. Joonis 3a kujutab olukorda, kus järjekordne täitmisele määratud μops asub trace cache´s (TC). Joonise 3b järgi toimib konveier siis, kui pöördumisel on toimunud TC möödalask, millele järgneb käsuvõtt teise taseme cache`st L2 ning selle dekodeerimine uopsideks ja salvestamine TC-sse. TC juurde kuulub ka keerukate IA-32 käskude mikroprogrammide püsimälu (ROM) (joonisel pole näidatud). Suurem osa programmist loetakse tavaliselt TC-st ja täidetakse ilma seisakuteta. Kaks programmiloendurit tagavad mõlemast programmilõngast sõltumatu pöördumise TC-sse ja seega mõlema lõnga sõltumatu täitmise. Kui mõlemad loogilised protsessorid üritavad üheaegselt pöörduda TC-sse, siis saavad nad seda teha ükshaaval üksteisele järgnevatel taktidel. Ühe loogilise protsessori seiskumise puhul saab teine pöörduda TC-sse igal taktil.
Pääs TC-sse on märgistatud lõnga informatsiooniga ja on vajaduse järgi dünaamiliselt muudetav. TC on 8-suunaline plokk-assotsiatiivne cache, millesse pääsu muutmine toimub LRU (Least-Resently Used -- kõige varem kasutatud (pääsu)) algoritmi järgi. Kuna mõlemad lõngad kasutavad TC-d ühiselt, siis vajaduse korral võib ühel loogilisel protsessoril olla rohkem pääsusid kui teisel.
Käsu puhul, mis sisaldab üle nelja μops'i, saadab TC mikrokäskude püsimällu keeruka käsu mikroprogrammi algaadressi, mille järgi püsimälu kontroller loeb püsimälust μopsid ja salvestab need TC-sse. Selleks, et mõlemad loogilised protsessorid saaksid sõltumatult täita IA-32 keerukate käskude mikroprogramme, on püsimälus kaks mikroprogrammi loendurit. Mõlema loogilise protsessori üheaegse pöördumise puhul püsimällu loevad protsessorid mikrokäske kordamööda üksteisele järgnevatel taktidel.
Kui toimub TC möödalask, siis loetakse käsk L2 cache-ist, dekodeeritakse uopsideks, mis seejärel salvestatakse TC-sse ja on sealt järgmistel pöördumistel kiiresti kättesaadav. Käsu lugemiseks L2-st saadetakse selle lineaaraadress käsuloendurist IP ITLB-sse (Instruction Translation Lookaside Buffer), kust saadakse käsu füüsiline aadress, millisel toimub lugemine. Loetud käsubaidid laaditakse kuni dekodeerimiseni puhvrisse. Kummagi loogilise protsessori jaoks on oma ITLB ja 64-baidine puhver.

Dekooder võtab puhvrist käsubaite ja dekodeerib need uopsideks. Kui mõlemad lõngad dekodeerivad käske üheaegselt, siis väljastavad kummagi lõnga puhvrid dekooderisse käske vaheldumisi. Kui ainult ühel lõngal on va TC möödalasu tõttu vajadus käsku dekodeerida, siis on kogu dekooderi ressurss selle käsutuses. Dekodeeritud käsud salvestatakse TC-sse ja edastatakse samaaegselt uopside järjekorda.
Pärast seda , kui uopsid on loetud TC-st või mikrokäskude ROM-ist või väljastatud käskude dekooderist, pannakse nad uopside järjekorda. Sellega lõpeb konveieri see osa, mis täidab käskusid saabumise järjekorras. Edasi võetakse uopse järjekorrast täitmiseks sellises järjestuses, millise määrab nende valmidus täitmiseks s.o. operandide olemasolu ja tehte sooritamiseks vaba plokk. See joonisel 4. kujutatud konveieri nn. "mittejärjekorras" täitmise osa koosneb ressursside paigutamise, registrite ümbernimetamise, plaanimise ja täitmise astmetest.
Paigutaja (allocator) võtab uopside järjekorrast uopse ning paneb paika puhvrid ja registrid mis on vajalikud iga uopsi täitmiseks. Selleks on 126 sisendiga ümbernimetamise puhver, 128 füüsilist registrit täisarvudega ning 128 füüsilist registrit ujukomaarvudega opereerimiseks, 48 laadimis- ja 24 salvestuspuhvrit. Mõned neist puhvritest on jaotatud nii, et kumbki loogiline protsessor saab kasutada poolt nende sisenditest.
Registrite ümbernimetamise loogika nimetab arhitektuursed IA-32 registrid ümber protsessori füüsilisteks registriteks. See lubab arhitektuurset IA-32 8-st registrist koosnevat täisarvude registrifaili töö käigus dünaamiliselt laiendada kuni 128 füüsilise registrini. Ümbernimetamise loogika kasutab registrite nimede muutmise tabelit (Register Alias Table – RAT), et jälgida arhitektuurse registri viimast asendusvarianti selleks, et teatada järgmisele käsule kust võtta lähteoperandid.
Kuna kumbki loogiline protsessor peab säilitama ja jälgima oma arhitektuurset olekut, on mõlemal oma RAT. Registrite ümbernimetamise protsess toimib üheaegselt ülalkirjeldatud paigutaja loogikaga. Seega töötab ümbernimetamise loogika sama uopsiga, millele paigaldaja parajasti eraldab ressursse.
Niipea, kui uopsidega on lõpetatud paigaldamise ja registrite ümbernimetamise protsessid, pannakse need kahte eraldi järjekorda: mäluoperatsioonid (laadimised ja salvestamised) ühte ja kõik ülejäänud operatsioonid teise. Neid kahte järjekorda nimetatakse vastavalt mälukäskude ja üldkäskude järjekorraks. Järjekorrad on jaotatud nii, et uopsid kummastki loogilisest protsessorist võivad hõivata kuni pooled mõlema järjekorra sisenditest.
Käskude plaanimine on uopside mittejärjekorras täitmise keskne protsess. Viis uopside plaanijat jaotavad erinevat tüüpi uopse nende tüübile täitmiseks vastavatesse sooritusplokkidesse. Koos võivad need plaanida kuni kuus uopsi igal protsessori taktil. Plaanijad määravad kindlaks, millal uopsid on

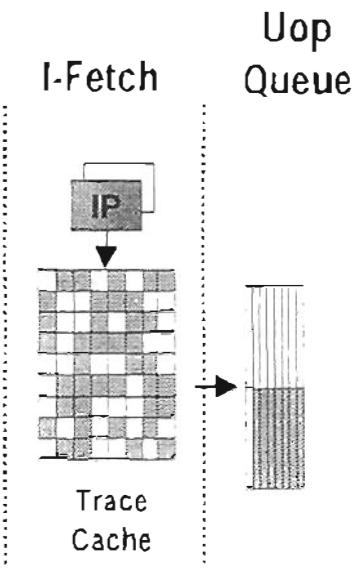
sooritusekс valmis sõltuvalt lähteoperandide ja vabade sooritusplokkide olemasolust.
Mälukäskude järjekorrast ja üldkäskude järjekorrast saadetakse uopse viie planeerija järjekordadesse nii kiiresti kui võimalik, valides seejuures igal taktil vaheldumisi mõlemale loogilisele protsessorile täitmiseks mõeldud uopse.
Igal plaanijal on kaheksa kuni kaheteistkümne sisendiga järjekord, millest see valib uopse sooritusplokkidesse. Plaanijad valivad uopse sõltumata sellest, kas need kuuluvad ühte või teise loogilisse protsessorisse (s.o. ühte või teise lõnga).
Tehete sooritusplokkidest koosnev protsessori füüsiline tuum ja mäluhierarhia jäävad loogilistele protsessoritele suures osas nähtamatuks. Kuna lähte- ja sihtregistrid olid varem ümbernimetatud füüsilisteks registriteks, siis valivad uopsid sihtregistritena ainult füüsilisi registreid laadides neisse tulemusi.



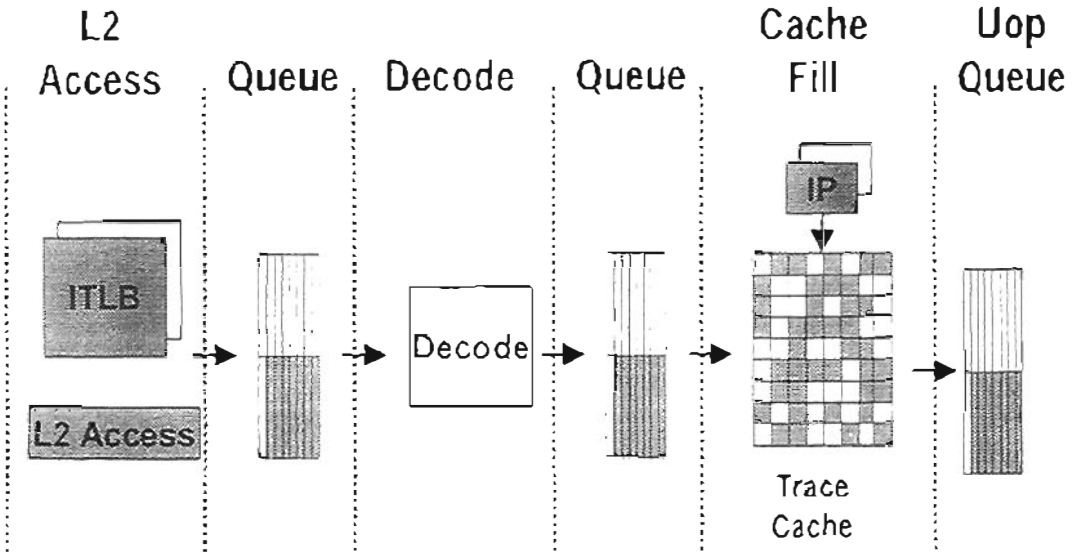
Joonis 1



Joonis 2

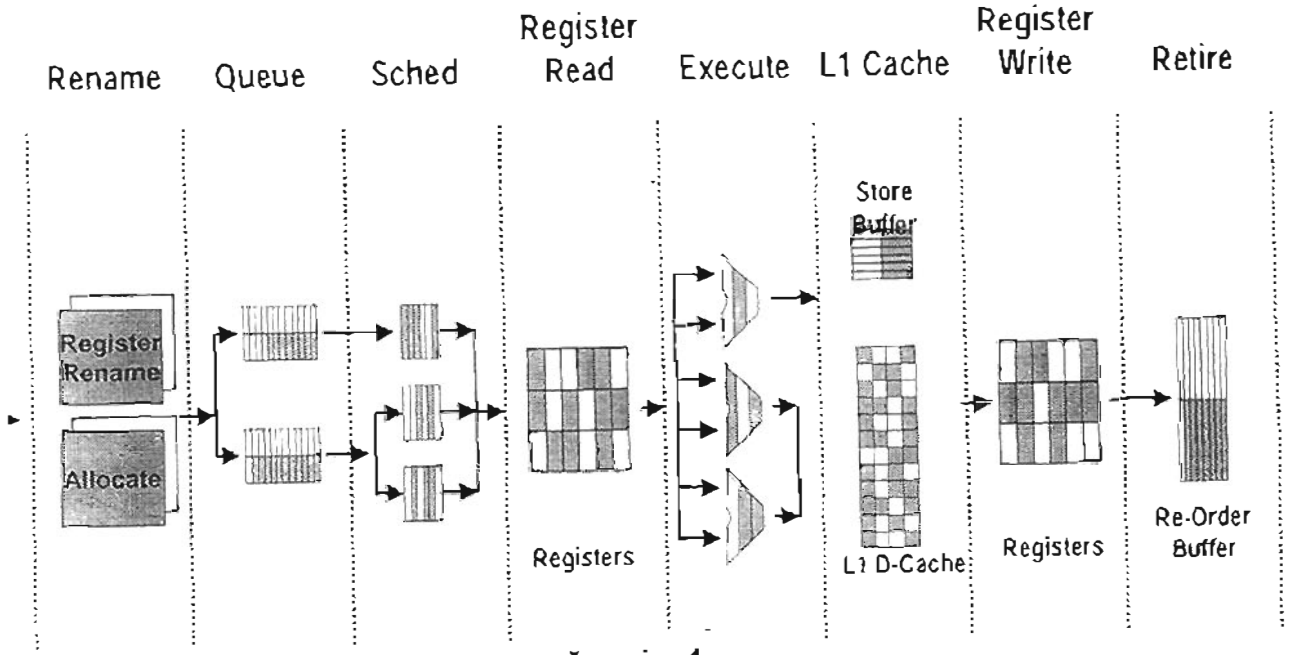


(a)



(b)

Joonis 3



Joonis 4

8. *Mälu juhtimine (Memory Management)*

8.1 *Protsessori kaitstud režiim*

Kaitstud (esialgse nimetusega virtuaal-) reziim realiseeriti esmaselt Inteli 16-bitises protsessoris 80286, mida esitleti 1982. aastal. Käesolevas loengumaterjalis käsitleme 1985 a. valminud 32-bitise protsessori 386 kaitstud reziimi. See on eelmise protsessori kaitstud reziimi laiendatud versioon, mis töötab kõigis järgnevates Inteli protsessorites (i486, Pentium, Pentium Pro, jne) ilma oluliste täiendusteta.

Kõige olulisemaks protsessori 386 kaitstud reziimi uuenduseks oli aparatuursel toel töötava mälu juhtimise süsteemi laiendamine. Mälu juhtimissüsteem tagab multitegumireziimis mälu otstarbeka jaotamise tegumite vahel , kiire lülitamise tegumilt tegumile , üksikutele tegumitele ja segmentidele kuuluvate mälupiirkondade effektiivse kaitse mäluruumis mahuga kuni 4 Gbaiti ja ühtlasi virtuaalmälu mahuga kuni 64 Tbaiti. Lisaks mälu segmentjaotusele omab 386 ka mälu jaotust lehekülgedeks ja nende kaitset samadel põhimõtetel nagu segmentjaotuseski.

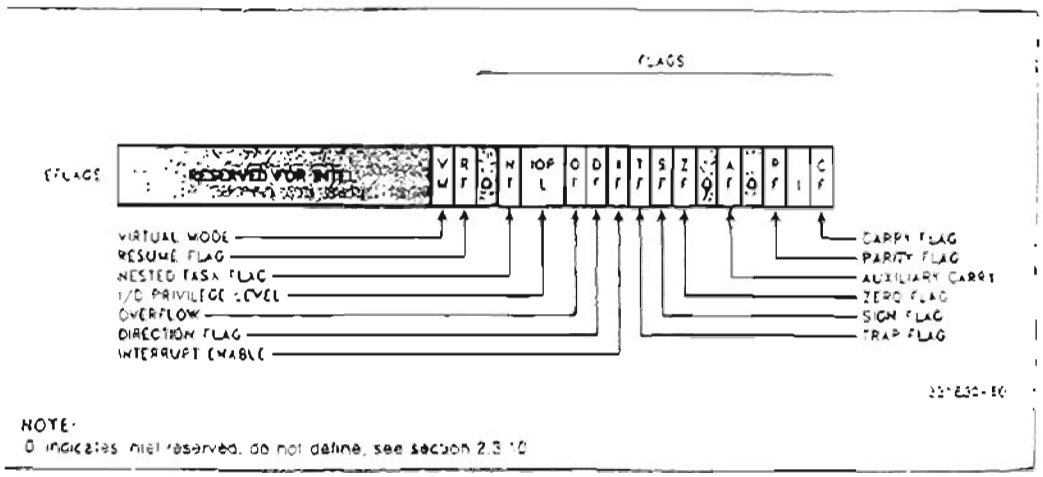
Protsessor 386 võib samuti nagu 286-ki töötada 8086 reziimis. Veelgi enam, protsessoril 386 põhineva arvuti käivitamisel hakkab see esmalt tööle 8086 reziimis, teostades alglaadimise , ja läheb alles seejärel automaatselt kaitstud reziimi. 8086 reziimi hakati 286-l ehitatud arvutites nimetama reaalreziimiks ja selles töötamise võimalus säilitati nii 386-s, kui ka kõigis järgnevates firma Intel protsessorimudelites.Reaalreziimis töötab 386 täpselt nii nagu 8086, adresseeritava füüsilise mälu mahuga kuni 1 Mbait, kuid märksa kiiremini ja võimalusega vajaduse korral kasutada 32-bitised registreid.

Protsessoris 386 on 6 segmentregistrit, lippuderegistrit on laiendatud 32 bitini, lisatud on 4 juhtregistrit (CR0, CR1, CR2 ja CR3) (joonis 1) , peale selle veel 8 veaavastus ja silumisregistrit (DR0...DR7) ning 2 testimisregistrit. Üldregistreid on endiselt kaheksa.

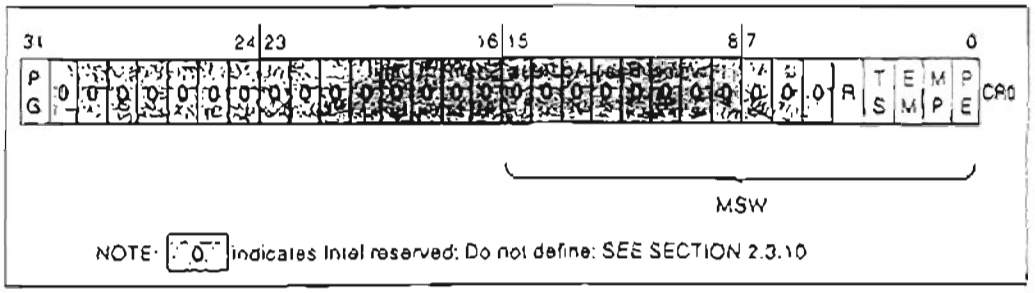
Võrreldes protsessoriga 8086 on protsessori 386 segmentregistrite sisu muutunud: segmendi baasi asemel laaditakse neisse segmendi poole pöördumisel segmendi selektor.

Protsessori poolt adresseeritav mäluruum jaguneb kaheks: globaalseks ja lokaalseks. Globaalses mäluruumis paiknevad kõikide tegumite poolt kasutatavad segmendid, lokaalne mäluruum aga jaguneb omakorda tegumite vahel privaatseteks piirkondadeks, millistes paiknevad eraldi iga üksiku tegumi segmendid.

Selektori abil mistahes segmendi poole pöördumine toimub kaudselt nn. segmendi deskriptori vahendusel. Iga segment omab 8- baidist (64 bitti) deskriptorit, mis sisaldab segmendi baasaadressi, segmendi suuruse piirväärtust ja nn. pääsuõigusi.

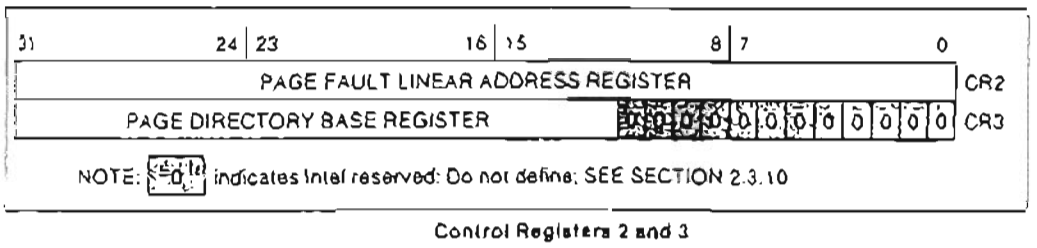


Flags Register



Control Register 0

PG – Paging Enable
R - Reserved
TS - Task Switched
EM- Emulate Coprocessor
MP – Monitor Coprocessor
PE - Protection Enable



Control Registers 2 and 3

Joonis 1

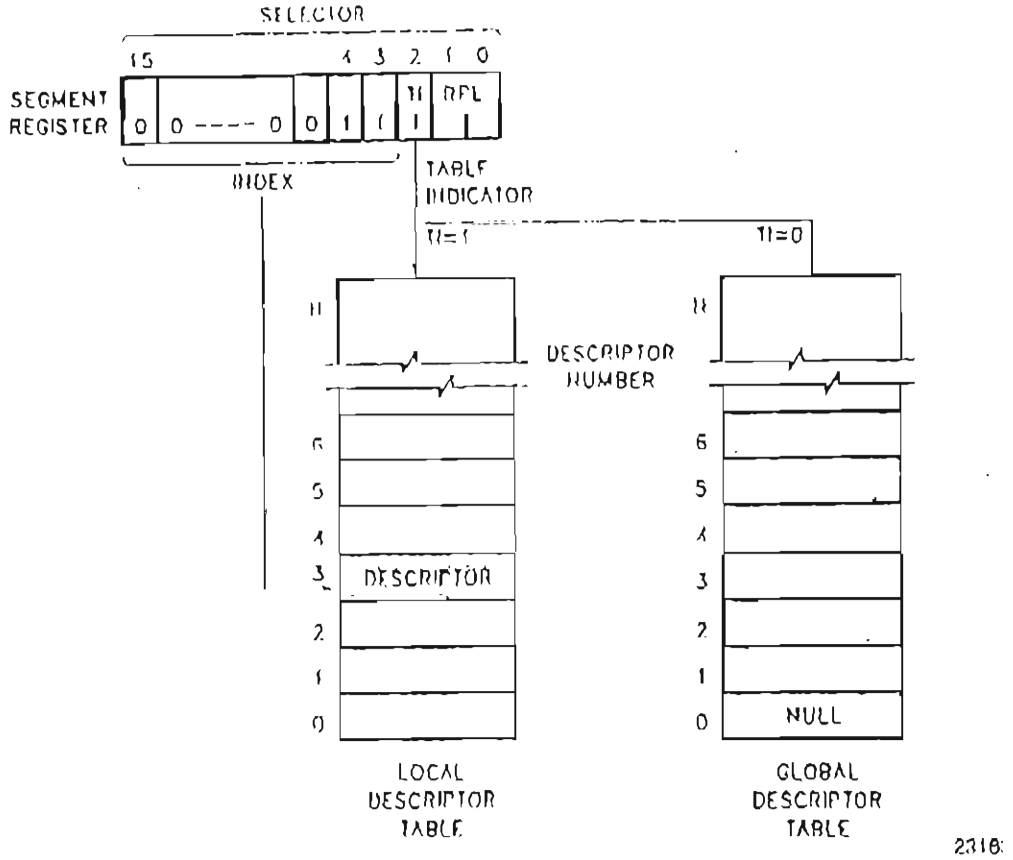
Kõikide segmentide deskriptorid on koondatud mälus paiknevatesse tabelitesse järgmiselt: globaalsegmentide deskriptorid ühte tabelisse (GDT), tegumite lokaalsegmentide deskriptorid iga tegumi oma lokaalsesse tabelisse (LDT). Segmendi selektorisse laaditava 13-bitise indeksi abil saab tabelis pöörduda kuni 8192 deskriptori poole (joonis 2). Segmendi kuuluvust kas globaalsesse või lokaalsesse mäluruumi tähistab selektoris sisalduva biti TI väärtus: kui TI=1, siis on segment lokaalne, kui aga TI=0, on segment globaalne.Kuna igal tegumil on juurdepääs globaalsetele segmentidele ja oma lokaasetele segmentidele, s.o. tabelitele GDT ja LDT, siis on iga tegumi käsutuses kuni 2^{13}ehk 16 384 segmenti. Segmendi maksimaalseks piirväärtuseks on 2^{32}ehk 4Gbaiti. Seega omab iga tegum kuni $2^{13} * 2^{32} = 64$ Tbaiti virtuaalmäluruumi, mis füüsiliselt moodustub põhimälu ja ketassalvesti koostöös.
Aparatuurselt tugineb 386 kaitstud reziimi töö järgmistel registritel:

globaaldeskriptorite tabeli register GDTR
lokaaldeskriptorite tabeli selektori register LDTR:
lokaaldeskriptortabeli deskriptori register :
katkestusdeskriptorite tabeli register IDTR:
tegumi register TR;
kuus segmentregistrit: CS, DS, SS, ES, FS ja GS
kuus segmenditeskriptori registrit:

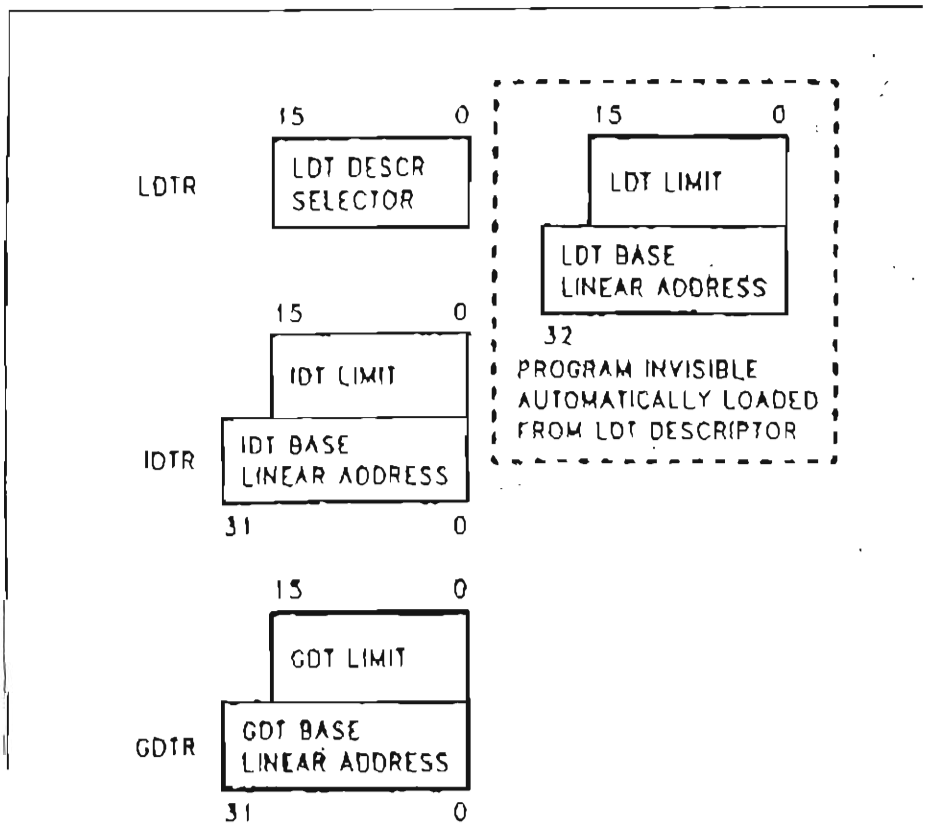
Loetletud registritest on lokaaltabeli deskriptori register ja kuus segmendideskriptorite registrit programmile nähtamatud (nn. vari-registrid) moodustates deskriptorite cache'i. Lokaaldeskriptortabeli deskriptori register laadub automaatselt LDTR laadimisel, segmendideskriptori register aga vastava segmentregistri (CS, DS, SS, ES, FS või GS) laadimisel.
Ülejäänud registreid saab laadida vastavate käskudega:

LGDT—Load Global Descriptor Table Register;
LLDT—Load Local Descriptor Table Register;
LIDT—Load Interrupt Descriptor Table Register;
LTR—Load Task Register;

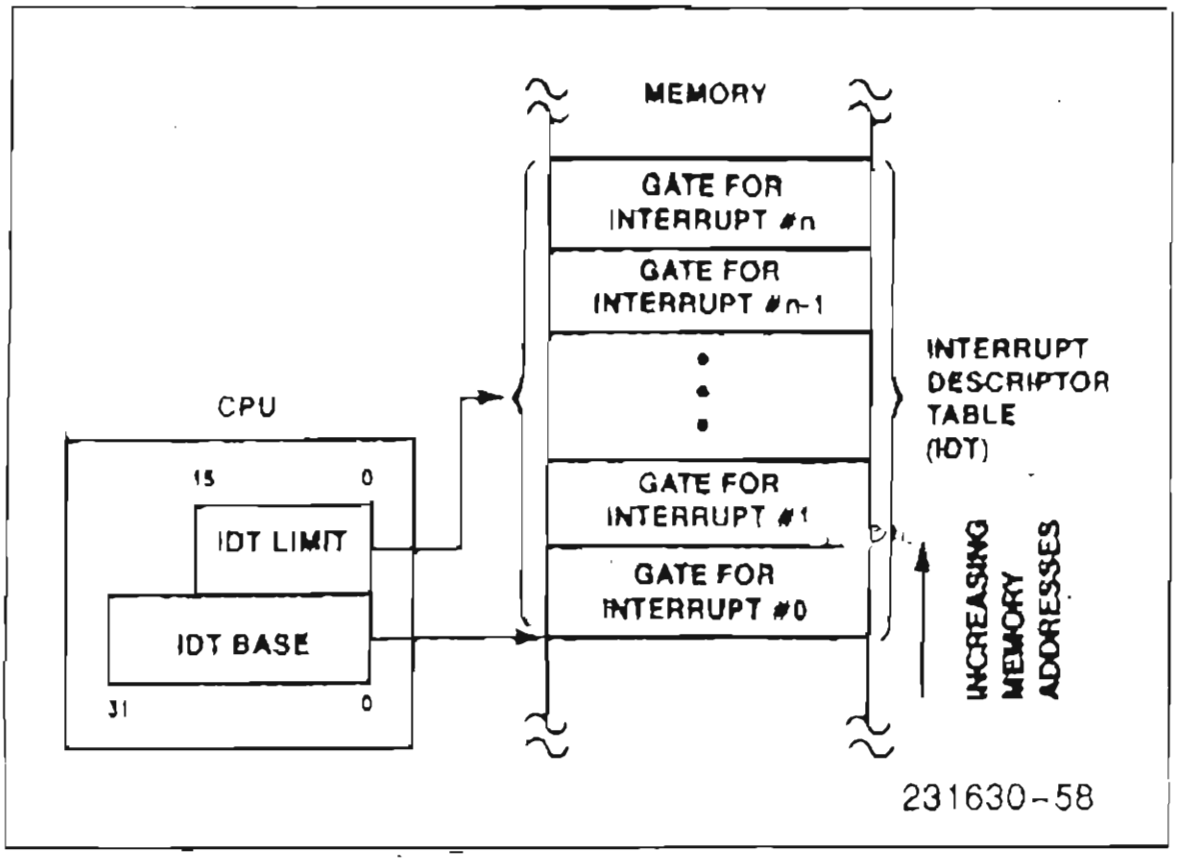
GDTR-s paikneb 32-bitine GDT baasi füüsiline aadress ja 16-bitine piirväärtus;
LDTR-s paikneb hetkel töötava tegumi LDT deskriptori 16-bitine selektor
(joonis 3);
IDTR-s paikneb 32-bitine IDT baasi füüsiline aadress ja 16-bitine piirväärtus
(joonis 4);



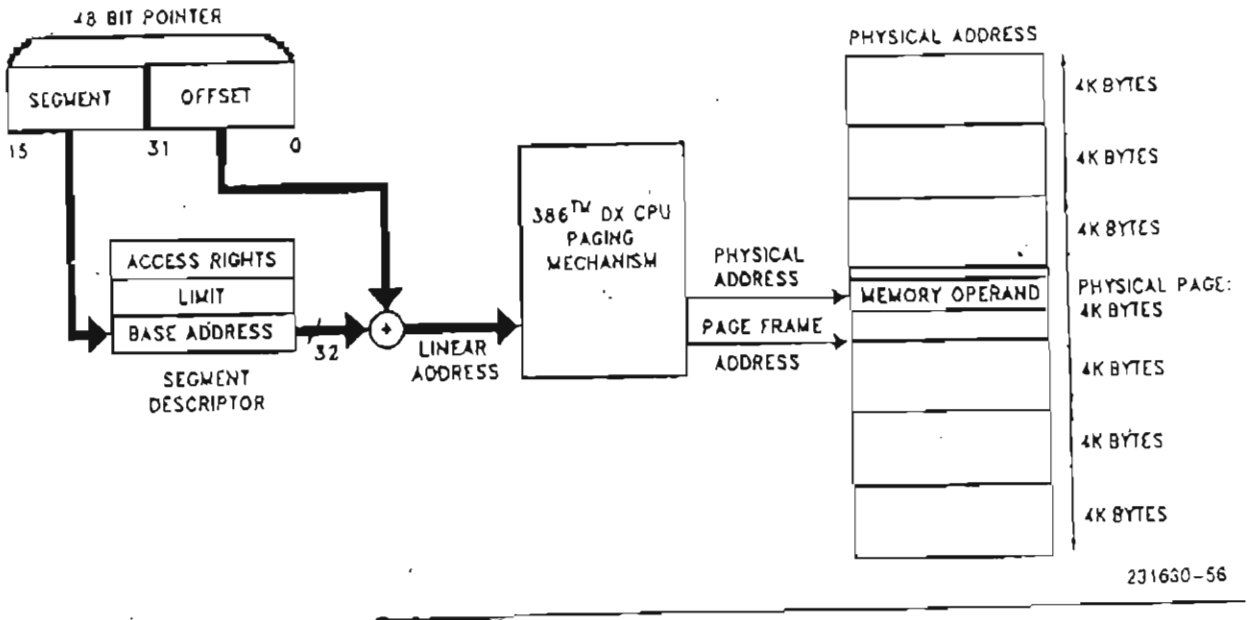
Joonis 2 Example Descriptor Selection



Joonis 3 **Descriptor Table Registers**



Joonis 4 **Interrupt Descriptor Table Register Use**



Joonis 5 Paging and Segmentation

Mällu pöördumine kaitstud reziimis toimub joonisel 5 kujutatud skeemi järgi. Loogiline aadress (skeemil 48-bitine POINTER) koosneb 16-bitisest segmendi selektorist ja 32-bitisest nihkest (OFFSET) segmendis. Selektori laadimisel segmentregistrisse, laadub vastav deskriptor automaatselt variregistrisse (joonis 6)

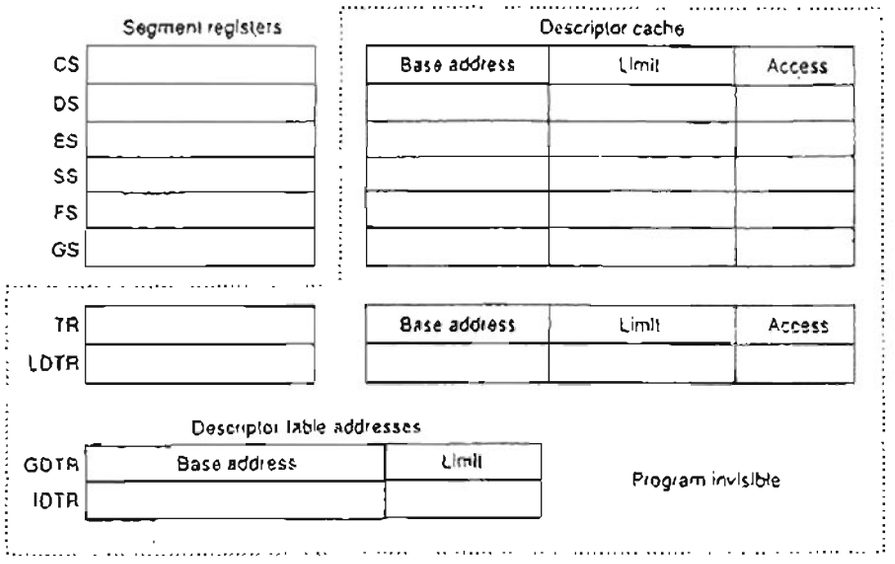
Deskriptoris olev segmendi 32-bitine baasaadress liidetakse loogilises aadressis toodud 32-bitise nihkega segmendis, mille tulemuseks on 32-bitine operandi füüsiline aadress. Samaaegselt füüsilise aadressi arvutusega toimub pääsuõiguste kontroll ja nihkeaadressi võrdlus deskriptoris antud 20-bitise piirväärtusega. Ülalkirjeldatust võib välja lugeda,et esmasel pöördumisel segmendi poole toimub kolm mälust lugemise tsüklit: käsu lugemine ja kaks tsüklit 8-baidise deskriptori lugemiseks tabelist 32-bitiste sõnade kaupa (automaatselt). Kõik järgnevad pöördumised sama segmendi poole ei nõua enam deskriptori lugemist, kuna viimane on vastavas protsessori variregistris juba olemas. Äsjakirjeldatud viisil toimub segmendi valik ja operandi aadressi arvutus juhul, kui selle deskriptor asub globaaltabelis (TI=0). Kui aga valitava segmendi deskriptor asub lokaaltabelis, siis oleks protsessi mõistmiseks vajalik alustada antud tegumile lülitumisest, sest kõne all on sel juhul multitegumreziim.

Pärast antud tegumile lülitumist laaditakse tegumi oleku segmendist(TSS) LDTR-sse selle tegumi lokaaldeskriptorite tabeli selektor. Seejärel toimub globaaltabelist seal paikneva lokaaltabeli (kui segmendi) deskritori automaatne lugemine ning laadimine lokaaltabeli deskriptori registrisse (variregister), mis vältab kaks lugemistsüklit.Selleks hetkeks on antud tegumi lokaaltabel valmis pöördumiste vastuvõtuks.

Kui nüüd toimub pöördumine segmendi poole, mille selektor viitab lokaaltabelisse (TI=1), siis liidetakse segmendi selektoris olev indeks (nihe lokaaltabelis) lokaaltabeli deskriptoris paikneva baasaadressiga . Tulemuseks on segmendi deskriptori aadress, millisel asuv deskriptor loetakse ning laaditakse automaatselt kahe lugemistsükliga segmendi - deskriptori registrisse. Deskriptoris olev segmendi 32-bitine baasaadress liidetakse loogilises aadressis toodud 32-bitise nihkega segmendis, mille tulemuseks on 32 bitine operandi füüsiline aadress.

Kõigi järgnevate pöördumiste puhul samasse segmenti, pole deskriptorit enam vaja lugeda , kuna see säilib variregistris seni kuni segmenti ei vahetata.

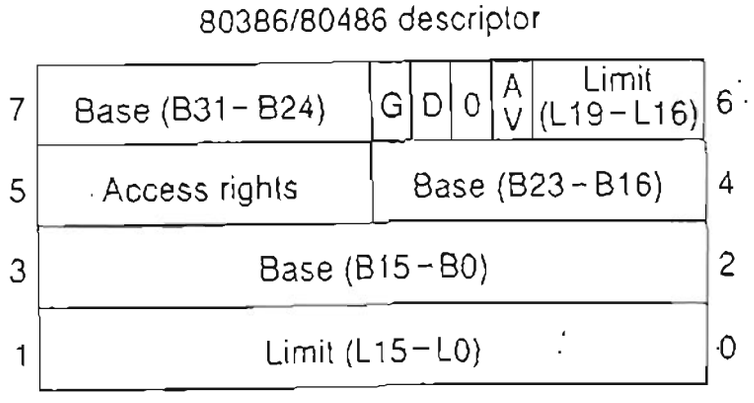
Segmendi deskriptor sisaldab peale segmendi baasaadressi (BASE), piirväärtuse(LIMIT) veel teisi segmenti iseloomustavaid atribuute , mis on koondatud nn. pääsuõiguste baiti. Programmi- ja andmesegmendi deskriptori formaat koos sisuseletusega on toodud joonisel 7.

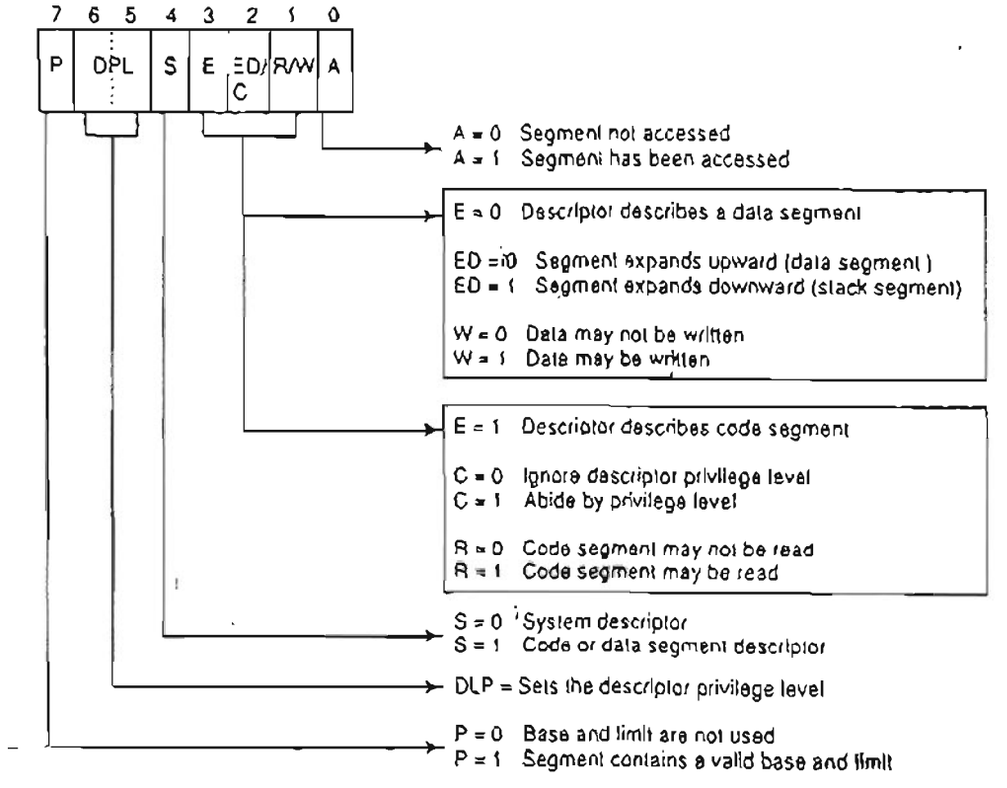


Notes:

1. The 80286 does not contain FS and GS or the program invisible portions of these registers
2. The 80286 contains a base address that is 24 bits and a limit that is 16 bits.
3. The 80386 and 80486 contain a base address that is 32 bits and a limit that is 20 bits.
4. The access rights are 8 bits in the 80286 and 12 bits in the 80386 and 80486.

Joonis 6 The program-invisible register within the 80286, 80386, and 80486 microprocessor.



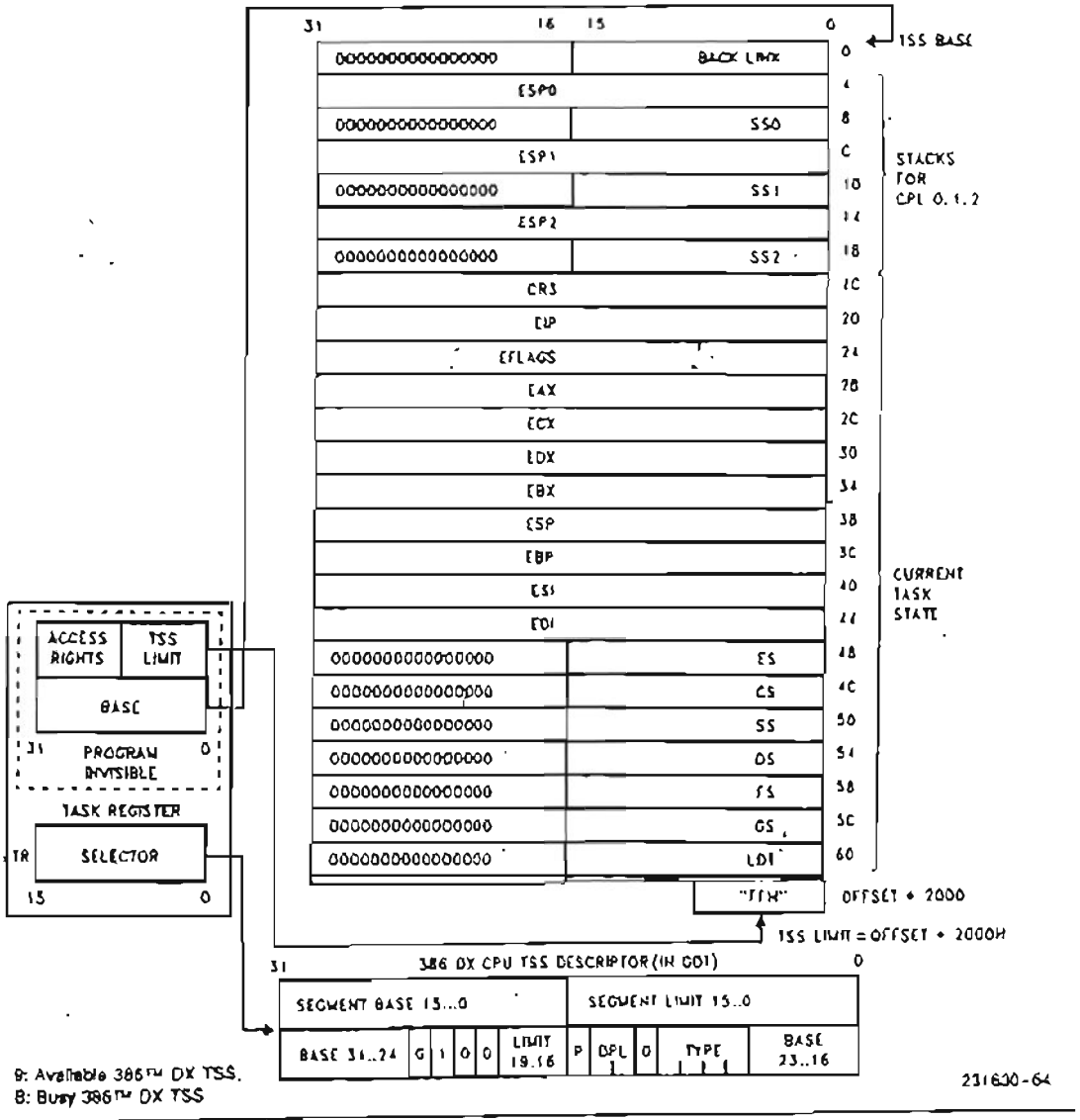


Joonis 7 The access rights byte for the 80286, 80386, and 80486 descriptor.

Pääsuõiguste baidi kolmebitine tüübi (TYPE) väli eristab üksteisest programmisegmendi ja andmesegmendi, määrates ära nende täidetavuse (Executable),salvestatavuse (Writeable) ja loetavuse (Readable). Bitt P näitab kas segment on füüsilises mälus või mitte. Kahebitine väli DPL (Descriptor Privilege Level) määrab deskriptori privileegi taseme. Bitt A näitab. kas segmendi poole on pöördutud või mitte. Bitt S eristab programmi- ja andmesegmendi deskriptorid kõigist teist tüüpi deskriptoritest .nagu süsteemsed - ja lüüsi deskriptorid.
Bittide P ja A vahendusel realiseerib operatsioonsüsteem virtuaalmälu s.o. kasutajale piiramatu mahuga malu illusiooni. Reaalselt on põhimälu maht piiratud ja seetõttu kõik tegumite segmendid alatiei mahu korraga põhimällu. Osa neist ollakse sel juhul sunnitud salvestama kettale. Kui antud hetkel vajaminev segment asub põhimälus, on bitt P=1, kui aga kettal, siis P=0. Viimasel juhul teostab operatsioonsüsteem vajaliku segmendi edastamise kettalt põhimällu. Kui põhimälus napib selleks ruumi, peab operatsioonsüsteem ruumi vabastamiseks eelnevalt mingi segmendi kettale salvestama. Kuid millise? Arvatavasti sellise, mida kaua pole kasutatud. Segmendi kasutamise sagedust registreeritakse biti A abil. Iga kord kui segmendi selektor laaditakse segmentregistrisse (mis tähendab antud segmendi järjekordset kasutamist) läheb bitt A olekusse 1.
Operatsioonsüsteem skaneerib perioodiliselt kõigi põhimälus olevate segmentide deskriptoreid kontrollides kõigi A bitte. Segmendi jaoks, mille A bitt oli olekus 1, lisab opsüsteem ajainkremendi, nullides seejuures biti A. Olekus null oleva A bitiga segmendile opsüsteem ajainkrementi ei lisa. Nii kogub operatsioonsüsteem informatsiooni segmentide kasutamise intensiivsuse kohta. Kui tekib küsimus, milline segment mälust kettale salvestamise teel kõrvaldada, siis kindlasti see mille summaarne kasutamise aeg on minimaalne. Seda on nimetatakse LRU (Least Resently Used) strateegiaks.
Multitegumrežiimi hõlpsamaks realiseerimiseks (s.o. ühelt tegumilt teisele ülemineku kiirendamiseks) on protsessoris tegumi register TR (Task Register) ja mälus iga tegumi jaoks tegumi oleku segment TSS (Task State Segment) (joonis 8).
Protsessori ümberlülitamine ühe tegumi täitmiselt teise tegumi täitmisele algab tegumi selektori laadimisega tegumi registrisse (TR) kaugsiirde (far jump) käsuga. Tegumi registrit saab laadida ka LTR käsuga või momendil täidetava tegumi TSS-ist. Tegumi selektor valib seejärel GTS-ist vastavava TSS-i deskriptori ja laalaadib selle variregistrisse.
Protsessor, olles tuvastanud, et TSS-i deskriptor on valitud, peatab jooksva tegumi täitmise selle olekusalvestamise teel TSS-i. Seejärel käivitab protsessor uue tegumi täitmise selle oleku laadimise teel valitud TSS-st protsessori registritesse. Protsessor hakkab uut tegumit täitma alates aadressist, mis on määratud uue tegumi TSS-ist protsessori laaditud CS:IP sisuga.

Lisaks veel selgitust tegumi mõiste täpsustamiseks. Kasutajaprogramm (application) koosneb ühest või enamast programmisegmendist ja grupist andmesegmentidest. Täitmise käigus peab programmil olema võimalus pöörduda ühe või enama programmi- või andmesegmendi aga samuti ühe või enama stackisegmendi poole. Kõik loetletud segmendid kokku võetuna moodustavadki operatsioonsüsteemi (OS) multitegumkeskonnas tegumi.

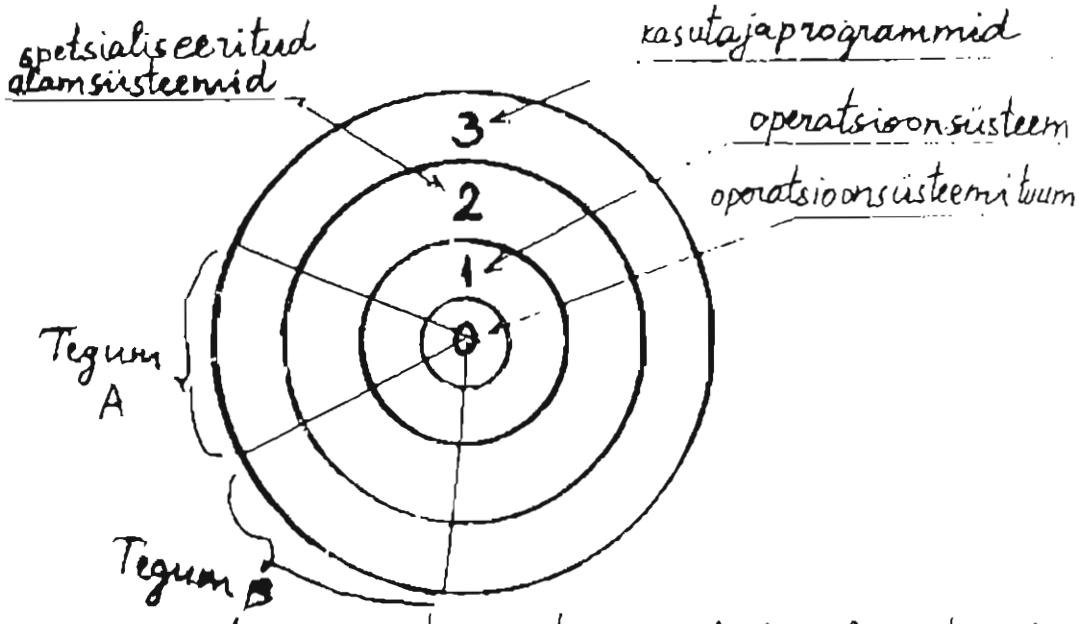
OS laadib kas kogu tegumi, või osa sellest põhimällu. Operatsioonsüsteem moodustab mälus ka andmestruktuuri, mis määrab protsessori oleku (s.o. täpse kujutise informatsioonist protsessori registrites) hetkel, mil see alustab (või jätkab) tegumi täitmist. Seda andmestruktuuri nimetataksegi tegumi oleku segmendiks (TSS). Multitegumkeskkonnas moodustab OS iga tegumi jaoks TSS-i. Lisaks moodustab OS globaaldeskriptortabelis (GDT) deskriptorid kõikide TSS-ide jaoks. TSS-i deskriptoris on kirjas selle baasaadress, pikkus ja privileegi tase (DPL).



Joonis 8 386™ DX TSS and TSS Registers

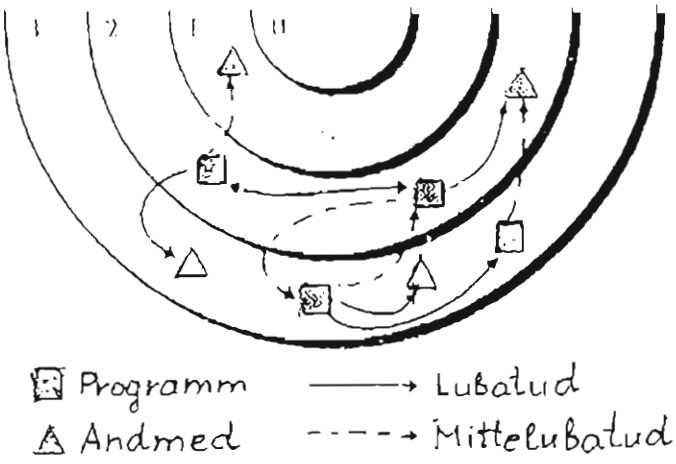
8.2 *Mälukaitse süsteem*

Mälukaitse on mitmeplaaniline ja toimib segmentide tasemel. Kaitse atribuudid paiknevad segmendi deskriptoris (vt. eelmise peatüki joon.7). Segmendi suuruse piirväärtus (LIMIT) väldib antud segmendi poole pöördumisel sattumast väljaspoole selle piire. Pääsuõiguste baidi segmendi tüüpi tähistavas väljas on rida bitte , millised kas lubavad või keelavad segmenti salvestamist ja sealt lugemist (bitid R ja W), liigitavad segmendid programmi- ja andmesegmentideks (bitt E), määravad segmendi laienemise suuna kas kasvavate või kahanevate aadressiväärtuste suunas (bitt ED) ja segmendi allvuse või mittealluvuse deskriptori privileegi tasemele (bitt C).Kahebitine privileegi taseme väli DPL (Descriptor Privileg Level) määrab deskriptori privileegi taseme. Bitt S eristab programmi- , andme- või stacki segmendi deskriptori süsteemisegmendi ja lüüsi deskriptorist.Bitt P näitab segmeni paiknemist põhimälus ja bitt A kajastab segmendi poole pöördumist.
Mälu kaitse privileegi tasemete järgi lubab kaitsta enam privilegeeritud (tähtsamaid) programme ja andmeid vähem privilegeeritud (vähem tähtsate) programmide poolse sanktsioneerimata sekkumise eest. Inteli protsessorite privileegisüsteem on neljatasemeline ja kodeeritud kahebitise väljaga. Segmendile omistatud privileegi taseme kahebitine kood Descriptor Privileg Level (DPL) paikneb deskriptori pääsuõiguste baidis. Momendil täidetava programmi privileegi taset nimetatakse jooksvaks privileegi tasemeks –Current Privileg Level (CPL) ja see on määratud selektoris CS asuva väljaga RPL (Requested Privileg Level). Juhtimise üleminekul teistsuguse privileegi tasemega programmile, laaditakse registrisse CS vastav selektor, mis on teistsuguse RPL väärtusega. CPL-I väärtust võib seega nimetada protsessori jooksvaks privileegi tasemeks.
Privileegi tasemeid on kujutatakse ükstese sees paiknevate kontsentriliste rõngastena , mis on alates keskmisest rõngast (ringist) nummerdatud 0-st kuni 3-ni (joonis 1), kusjuures 0 on kõrgeim privleegi tase, sellele järgnevad alanemise suunas tasemed 1 ja 2, ning 3 tähistab madalaimat privileegitaset. Segment, mille deskriptoris toodud DPL = 0, on enim kaitstud.Juurdepääs sellele on vaid programmidel, mis asuvad samal 0 tasemel. Segment, mille deskriptoris DPL=3 on vähim kaitstud. Sellele segmendile on avatud juurdepääs kõigilt privileegitasemetelt.
0-tasemel paikneb operatsioonsüsteemi tuum, mis hõlmab seda osa operatsioonsüsteemist, mis realiseerib mälu juhtimist ja kaitset;
1-tasemel asub ülejäänud osa opeeratsioonsüsteemist;
2-tase on ette nähtud spetsialiseeritud alamsüsteemidele,nagu andmebaasid. programmeerimissüsteemid jne.;



0 – operatsioonsüsteemi tuum: hõlmab seda või seda osa operatsioonsüsteemist, mis realiseerib mälu juhtimist, mälu kaitset ja juurdepääsu.
1 – operatsioonsüsteem, sisend/väljundsüsteem.
2 – spetsialiseeritud alamsüsteemid: näit. andmebaaside juhtimissüsteem, programmide koostamise süsteemid jne.
3 – kasutajaprogrammid

Joonis 1



Joonis 2

3-tase on kasutajaprogrammide tase;
Andmete poole pöördumisel ei tohi jooksva programmisegmendi deskriptori DPLi (s.o. CPLi) arvväärtus olla suurem andmesegmendi deskriptori DPLi arvväärtusest, mis tähendab, et programm ei või olla vähem privilegeeritud, kui andmed, mille poole ta pöördub:

$$CPL <= DPL$$

Selle tingimuse täitmist kontrollitakse selektori laadimisel ühte segmentregistritest DS või ES. Tingimuse mittetäitmisel selektorit ei laadita vaid formeeritakse üldise kaitse rikkumise erijuhtum (katkestus 13). Selektori laadimisel stacki segmentregistrisse kehtib rangem tingimus: laadimine lubatakse vaid sel juhul, kui stacki segmendi DPL arvväärtus võrdub CPLi arvväärtusega. Teisiti öeldes, jooksev programm ei saa kasutada ka madalama privileegi tasemega stacki, rääkimata kõrgemast tasemest. Peale selle SSi selektori laadimiseks, peab valitavasse segmenti olema võimalik nii salvestus, kui ka sealt lugemine. Lisaks peab segment olema mälus (bitt P=1).

Programmide privilegeeritud kaitse on rangem kui andmete kaitse: antud privileegi tasemel olevast programmist saab siirduda vaid samal privileegi tasemel olevasse programmisegmenti (kasutades käske CALL ja JMP). See tähendab, et CPL väärtus peab võrduma sihtsegmendi DPL-ga. Muidugi peab see reegel lubama erandeid, sest vastasel juhul ei saaks taseme 3 rakendusprogrammid kasutada operatsioonsüsteemi poolt pakutavaid teenindusprogramme, mis asuvad tasemetel 1 ja 0.
Kaitserõngaste ulatuses lubatud ja mittelubatud pöördumised ja siirded programmidest võtab kokku joonis 2.
Üks võimalusi siirdumiseks antud privileegi tasemega programmist kõrgema privileegi tasemega programmi on nn. alluvate programmisegmentide kasutamine. Selleks, et muuta mingi programmisegment tema poole pöörduva segmendi suhtes alluvaks, tuleb selle segmendi pääsuõiguste baidis asuv alluvuse bitt C panna olekusse 1. Alluvasse segmenti saab teostada siiret käskudega CALL või JMP ka madalama privileegi tasemega programmidest. Teiste sõnadega, alluv programmisegment tema poole pöörduva programmi privileegi taset ei kontrolli, vaid hakkab seda täitma pöörduva programmi privileegi tasemel. Näiteks, kui see on 3, siis hakkab tööle CPL=3, kui aga see on 0, siis jooksval tasemel CPL=0.
Alluv segment, omades sellist "liikuvat" privileegi taset, muutub praktiliselt kaitsetuks ja seetõttu ei või selles kasutada privilegeeritud käskusid (vt. vastavat lõiku), nagu näiteks sisestamis- väljastamiskäsud. Ainus piirang, mis kehtib alluva segmendi suhtes, on see , et selle

DPL väärtus peab olema väiksem või võrdne jooksva CPL väärtusega võrreldes. Teiste sõnadega, alluvuse bitti kasutades saab teostada siiret sisemistesse , enam kaitstud rõngastesse. Vastassuunas on niikuinii vaba pääs.
Kaitstud siire kõrgemal privileegitasemel olevasse programmi saab toimuda vaid eriliste juhtimisdeskriptorite nn. lüüside (gates) abil , milliseid võib paigutada deskriptortabelitesse. Protsessori kaitsesüsteemis on neli erinevat tüüpi lüüsi: väljakutse lüüs (call gate). katkestuse lüüs (interrupt gate). tegumi lüüs (task gate) ja püünise lüüs (trap gate). Joonisel 3 on kujutatud lüüsi deskriptori üldformaat ja toodud tabel , mis kirjeldab kõigi nelja tüüpi lüüsi välju.Väljakutse lüüsi deskriptor on "vahendajaks"erinevatel privileegitasemetel olevate programmisegmentide vahel . Väljakutse lüüs identifitseerib sisenemispunkti kõrgemal privileegitasemel olevasse programmi. Teiste sõnadega, väljakutse lüüs adresseerib protseduuri (alamprogrammi), mis asub kõrgema privileegitasemega segmendis ja ei ole seetõttu otseselt kättesaadav.
Selektorit, mis valib deskriptortabelist lüüsi deskriptori tohib laadida vaid programmisegmendi registrisse CS. Väljakutse lüüsi saab adresseerida ainult kaugväljakutse käsuga CALL (JMP käsu kasutamine on keelatud). CALL käsus kasutatakse vaid selektori välja ,mis viitab väljakutse lüüsile. Lüüsis sisalduv selektor viitab omakorda vajalikku protseduuri sisaldava programmisegmendi deskriptorile. Lüüsis toodud nihkeaadress määrab sisendpunkti, mis sisuliselt on protseduuri algpunkt. Tulemuseks on protseduuri käivitumine õigest kohast (joonis 4).
Väljakutse lüüs, olles vahelüliks programse juhtimise üleandmisel kõrgema privileegi tasemele, omab ka ise teatavaid kasutuse kitsendusi. Lüüs, kui deskriptor omab samuti privileegi välja DPL ja võib seega asetseda erinevatel privileegi tasemetel. Lüüsi poole pöördumisel tuleb arvesse võtta järgmisi privileegitaseme väärtusi:
--väljakutselüüsi enda DPLi väärtus;
--väljakutsutava (siht) programmisegmendi deskriptori DPLi väärtus;
--kaugväljakutse käsu CALL selektorväljas olev RPLi väärtus:
--väljakutsuva (jooksva) programmi CPLi väärtus;
Lüüsi kaudu väljakutse lubamise tingimus on üldkujul järgmine:

sihtprogrammi DPL<= max(RPL, CPL) <= lüüsi DPL

. Kui näiteks protsessor täidab taseme 2 programmi (CPL=2) ja sellel on vaja välja kutsuda taseme 0 protseduuri (siht DPL=0),siis tuleb kasutada kas 2 või 3 tasemel olevat lüüsi (lüüsi DPL= 2 või 3). Kui väljakutse lüüsi DPL=1, siis on lüüs kätte saadav programmisegmentidele, mille DPL=0 või 1, aga programmisegmentidele, mille DPL=2 või 3,

31 24	16	8	5	0	
SELECTOR	OFFSET 15 . . . 0				0
OFFSET 31 . . . 16	P	DPL	0	TYPE	0 0 0 WORD COUNT 4 . . . 0	+4

Gate Descriptor Fields

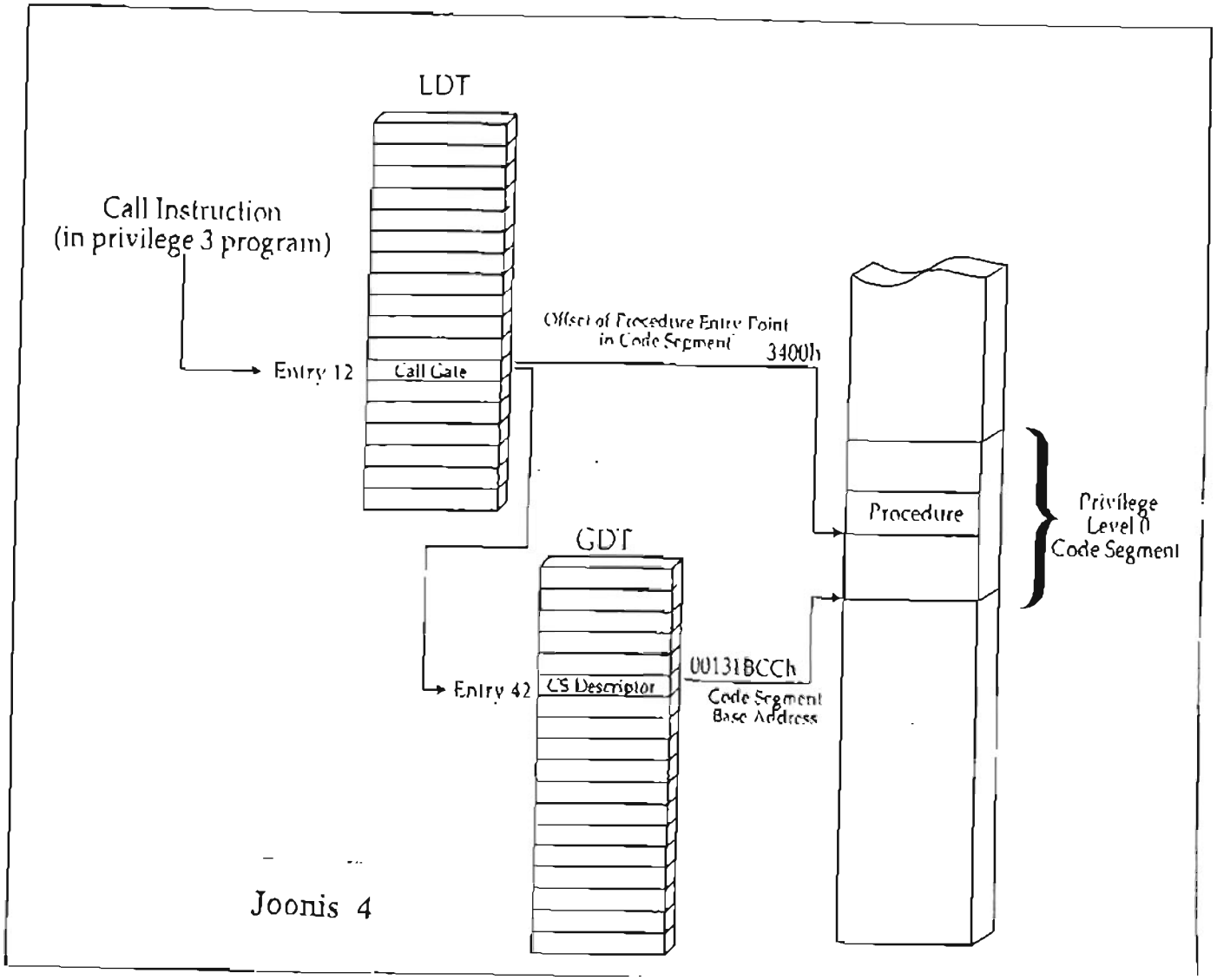
Name	Value	Description
Type	4	80286 call gate
	5	Task gate (for 80286 or 386™ DX task)
	6	80286 interrupt gate
	7	80286 trap gate
	C	386™ DX call gate
	E	386™ DX interrupt gate
	F	386™ DX trap gate
P	0	Descriptor contents are not valid
	1	Descriptor contents are valid

DPL—least privileged level at which a task may access the gate. WORD COUNT 0-31—the number of parameters to copy from caller's stack to the called procedure's stack. The parameters are 32-bit quantities for 386™ DX gates, and 16-bit quantities for 80286 gates.

DESTINATION SELECTOR	16-bit selector	Selector to the target code segment or Selector to the target task state segment for task gate
DESTINATION OFFSET	offset 16-bit 80286 32-bit 386™ DX	Entry point within the target code segment

Joonis 3 Gate Descriptor Formats

Call Gate and CS Descriptors, Code Segment and Called Procedure



Joonis 4

kättesaamatu. Mõningad lubatud siirdumise variandid läbi väljakutse lüüsi on toodud joonisel 5.
Tuleb märkida, ülalkirjeldatud lüüsi privileegi taseme kontrolli reeglid lubavad siirduda protseduuri läbi lüüsi, mille DPL väärtus võrdub jooksva programmi privileegi taseme (CPL) väärtusega. Kuid see on asjatu vaev, sest sama taseme protseduuri saab siirduda CALL käsuga otse, ilma lüüsi kasutamata.

Stacki ümberlülitamine. Ülal oli mainitud, et programm saab kasutada ainult samal privileegi tasemel olevat stacki segmenti. Et seda nõuet mitte rikkuda , protsessor minnes üle kõrgema privileegi tasemega programmi täitmisele lülitab automaatselt stacki ümber protsessoriga võrdse privileegiga stacki segmenti. Selleks otstarbeks on iga tegumi oleku segmendis (TSS) viidad (SS:SP)kolme erineva privileegitasemega (0, 1 ja 2) stacki segmenti. Taseme 3 jaoks pole viita vaja , sest madalamat privleegitaset kui 3, ei esine.
Pärast edukat CALL käsuga väljakutse lüüsi läbimist ilma privilegeeritud kaitse reegleid rikkumata, peab protsessor jätkama tööd uue stackiga.Selleks laaditakse registrid SS ja SP uue taseme stacki viidaga TSSist. Kui näiteks oli välja kutsutud taseme 1 protseduur siis laaditakse SS1 ja SP1. Seejärel salvestab protsessor uude stacki vana stacki tipu viida, s.o. SS ja SP sisud. Edasi analüüsib protsessor lüüsi deskriptori välja WORD COUNT ning kopeerib automaatselt vanast stackist selles näidatud arvu sõnu. Need kujutavad endast parameetreid , mis tuli edastada protseduuri. Lõpuks salvestatakse uude stacki programmi naasmise aadress (CS:IP sisu).
Väljakutsutud protseduuri initsialiseerimiseks jääb üle vaid laadida lüüsi deskriptoris toodud selektori ja nihke väärtused registritesse CS ja IP ning teostada privilegeeritud kaitse kontroll. Eduka kontrolli järel täidab protseduur oma funktsioonid. Täitmise käigus võib protseduur pöörduda kõikidesse vähem privilegeeritud segmentidesse, sealhulgas ka protseduuri väljakutsunud programmi stacki ja andmesegmenti. Näiteks suurte parameetriplokkide edastamiseks võib väljakutsuv programm edastada protseduurile nende plokkide viidad ja parameetrite arvu, misjärel võib protseduur neid kaudselt ise lugeda.
Pärast protseduuri täitmist toimub naasmine programmi käskudega RET või RETn . Käsk RETn kõrvaldab stackist n sõna ja on vajalik sel juhul, kui väljakutsel edastati parameetrid.

Privilegeeritud käsud. Protsessoril on rida privilegeeritud käske, mille privilegeerituse tase on erinev. Käskusid, mis mõjutavad segmenteerimise ja kaitse mehhanisme, võib kasutada ainult taseme 0 programmides:

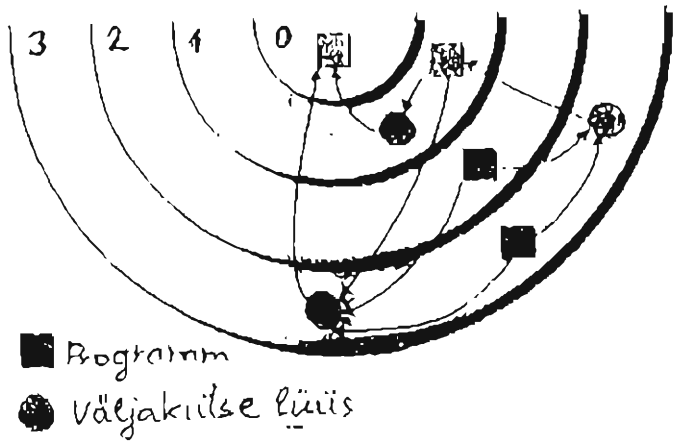
```
HLT                 ;protsessori seiskamine
```

CLTS	;tegumi ümberlülimise lipu nullimine
LGDT, LIDT, LLDT	; deskriptortabelite registrite laadimine
LTR	; tegumi registri laadimirle
LMSW	; masina olekusõna laadimine

Teise gruppi moodustavad käsud, mis muudavad katkestuste lipu IF olekut ja teostavad andmete saisestust- väljastust.

CLI	;katkestuste keelamine
STI	;katkestuste lubamine
IN, INS	;andmete sisestamine pordist
OUT, OUTS	;andmete väljastamine porti

Nende käskude täitmiseks ei pea programm olema tingimata tasemel 0. Nende käskude privilegeeritus seisneb selles, et neid võivad täita programmid, mille privileegi tase on kõrgem lippuderegistri FLAGS väljaga IOPL määratud sisestus- väljastus privileegitasemest.Teiste sõnadega, nende käskude täitmiseks peab CPL olema arvuliselt väiksem või võrdne IOPLi arvväärtusega võrreldes. Näiteks, kui IOPL=3, siis võivad teise grupi käskusid täita kõikide tasemete programmid, kui aga IOPL=0, on nad täidetavad ainult taseme o programmide poolt. Sisestus-väljastuse privileegi taset võivad muuta vaid taseme 0 prorrammid. Katkestuste lipu IF oleku muutmiseks peab programmile olema lubatud sisestus-väljastus (CPL<= IOPL).



Joonis 5

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