1. Sissejuhatus

von Neumanni arhitektuur:

1. mälu programmi koodi ja andmete hoidmiseks;
2. juhtseade käskude dekodeerimiseks;
3. aritmeetikaseade aritmeetikaoperatsioonide teostamiseks;
4. sisend-väljund liidesed perifeerseadmete ühendamiseks ja andmete sisestamiseks.

Juhtseade ja aritmeetikaseade moodustavad CPU. Mälu koosneb üldjuhul adresseeritavast põhimälust ja mahukamast, kuid tunduvalt aeglasemast ketasmälust. Mälus hoitakse nii andmeid kui ka programme. Kui kasutatakse kahte eraldi mälu andmete ja programmide hoidmiseks, on tegemist Harvardi arhitektuuriga. CPUs on registrid andmete, vahetulemuste või juhtinformatsiooni hoidmiseks. Registrite arv sõltub suuresti konkreetsest arhitektuurist, kuid alati on olemas käsuloendur (IP, PC) ja pinuviit (SP). Andmete esitamiseks kasutatakse kahendkoodi. Käsud esitavad tavaliselt üsna lihtsaid aritmeetika- ja loogikaoperatsioone ja keerulisemad käsud saadakse nende kombineerimisel.

Von Neumanni arhitektuuri põhijooned:

1.käsud esitavad lihtsaid aritmeetika- ja loogikaoperatsioone;

2.kasutaja (programmeerija) valib käskude kogumist välja käsud vajalikus järjestuses ja ühendab need programmiks;

3.mällu salvestatud käskude jada (programm) täidetakse järjestikuliselt;

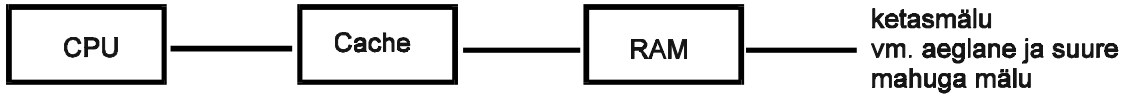
4.käsud ja andmed on tavaliselt salvestatud ühte ja samasse mällu.

Põhimõtteliselt täidab protsessor kahte komplekset operatsiooni:

- käsu lugemine mälust;
- käsu täitmine.

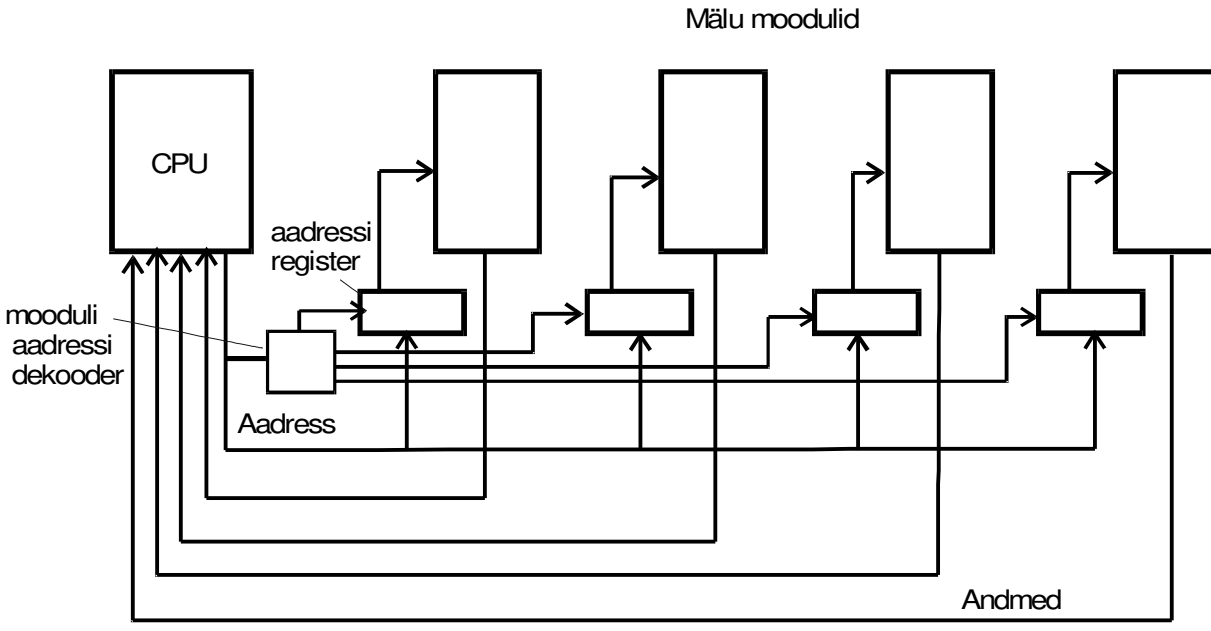
käsu lugemisele järgneb käsu dekodeerimine. Ka käsu täitmise saab jagada mitmeks erinevaks tegevuseks (näiteks esimese operandi lugemine mälust, teise operandi lugemine, aritmeetika- või loogikaoperatsiooni täitmine nende operandidega). Kui eelmise ja järgmise operatsiooni täitmine mingil määral kattuvad, nimetatakse seda ülekatteks (*fetch/execute overlap*). Selleks, et käske saaks täita ülekattega, kasutatakse peaaegu kõikides kaasaegsetes mikroprotsessorites konveieried (*pipelines*).

Enam-vähem kõigi arhitektuuride puhul töötleb protsessor informatsiooni tunduvalt kiiremini, kui toimub mälu poole pöördumine. Et protsessori ja mälu kiirusi enam-vähem ühildada, kasutatakse hierarhlist mälu ülesehitust.



1.1. Memory interleaving

Kõige lihtsam viis suurendada aeglase seadme jõudlust on selliste seadmete arvu suurendada ja panna nad paralleelselt tööle. Mälu puhul nimetatakse sellist arhitektuurilist lahendust *memory interleaving*. Kui mälu on protsessorist *n* korda aeglasem, siis pannes tööle *n* mälu moodulit ja pöördudes nende kõigi poole korraga, saavutatakse protsessoriga võrdne kiirus. Kõige lihtsamal juhul ühendab protsessorit mäludega ainult üks aadresssiin ja kõigist *n* mäluelemendist loetakse andmed korraga ühelt ja samalt aadressilt, ehk teisiti öeldes on lihtsalt tegemist *n* korda pikema sõnaga. Teisel juhul jaguneb mälu aadressväli kahte ossa: esimest osa kasutatakse mälu mooduli valimiseks ja teine osa määrab ära aadressi mälu mooduli sees.



2. Konveierid (*pipelines*)

Protsessor täidab põhimõtteliselt kahte erinevat operatsiooni:

1. käsu lugemine mälust
2. käsu täitmine

Veelgi täpsemalt jaotades saame kolm põhilist operatsiooni:

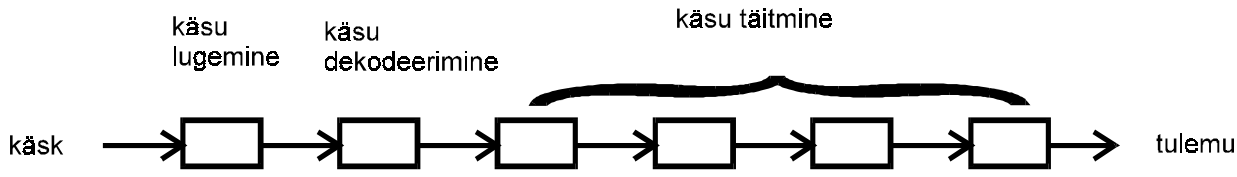
1. käsu lugemine mälust
2. käsu dekodeerimine
3. käsu täitmine

Niimoodi jaotust täpsustades saame veelgi enam erinevaid operatsioone. Näiteks 5 erinevat:

1. käsu lugemine
2. käsu dekodeerimine
3. operandide lugemine mälust
4. operatsiooni täitmine

5. tulemuse salvestamine

Konveieri puhul täidab igat operatsiooni erinev seade ja iga seadme sisend on ühendatud järgmise väljundiga.



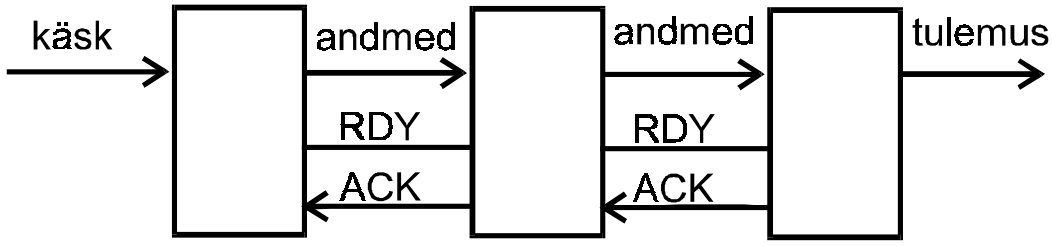
Tulemus saadakse konveieri kõige viimase seadme (mõnikord ka vahepealse seadme) väljundist. Käskude täitmise summaarne aeg konveieril lüheneb, kui täidetakse järjestikku mitu ühesugust käsku.

Konveiereid liigitatakse:

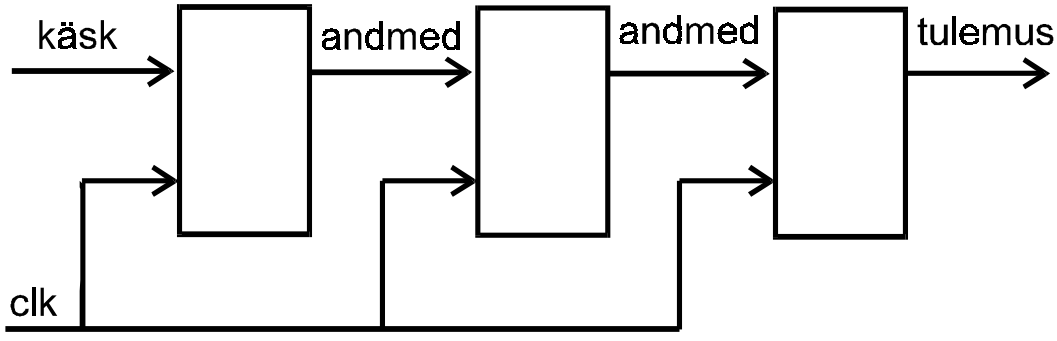
- asünkroonsed konveierid
- sünkroonsed konveierid

Asünkroonsete konveierite puhul lisatakse konveieri osade vahele kviteerimissignaalid “ready” ja “acknowledge”. “ready” signaal informeerib konveieri järgmist seadet sellest, et operatsiooni täitmine on lõpetatud ja andmed on valmis saatmiseks järgmisesse seadmesse. Järgmine seade vastab “acknowledge” signaaliga kui ta on valmis andmeid vastu võtma. Sünkroonse konveieri puhul edastatakse andmed sünkrosignaaliga kindlate intervallide järel.

asünkroonne konveier:



sünkroonne konveier:



Tavaliselt kasutatakse sünkroonse konveieri puhul puhvreid erinevate seadmete vahel ja sünkrosignaal on registritele lubavaks signaaliks. Kuid konveierid võivad olla ka ilma sünkrosignaalita ja registriteta. Tulemus edastatakse siis järgmisele seadmele kohe kui ta jõuab eelmise seadme väljundisse. Selline konveier töötab maksimaalse võimaliku kiirusega, kuid tema projekteerimine ja testimine on väga keeruline.

Tavaliselt ei võta kõikide operatsioonide täitmine konveieris täpselt ühe palju aega. Jaotades operatsioonide täitmise suurema arvu seadmete vahel püütakse nende töökiirust mõnevõrra ühtlustada ja summaarset töökiirust suurendada. Sünkrosignaali sagedus sõltub konveieri kõige aeglsema osa töökiirusest.

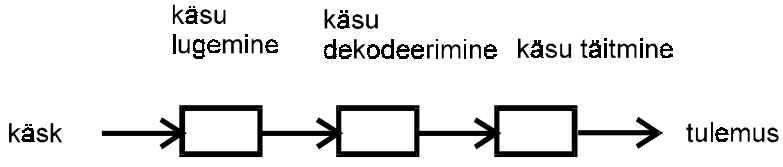
Konveieri töö aeglustub ja projekteerimine raskeneb põhiliselt kolmel põhjusel:

1. konfliktid riistvaraliste ressursside nõudmisel;
2. andmetevaheline sõltuvus;
3. hargnemiskäsud.

2.1. Konfliktid ressursside nõudmisel

Kõige sagedamini tekib konflikt, kui mitu erinevat konveieri osa pöörduvad korraga mälu poole (näiteks üks käsu lugemiseks ja teine operandide lugemiseks).

Olgu konveier kolmeosaline:



aeg

t1	1.käsu lugemine		
t2	2.käsu lugemine	1.käsu dekodeerimine	
t3	3.käsu lugemine	2.käsu dekodeerimine	1.käsu operandide lugemine

Ajahetkel t3 pöörduvad nii esimene kui ka kolmas seade mälu poole, et sealt lugeda. Probleem laheneb, kui kasutatakse mitut erinevat mälumoodulit, siis näiteks 3.käsk ja 1.käsu operandid asuvad erinevates mälu moodulites ja konflikti ei teki.

2.2. Hargnemiskäsud

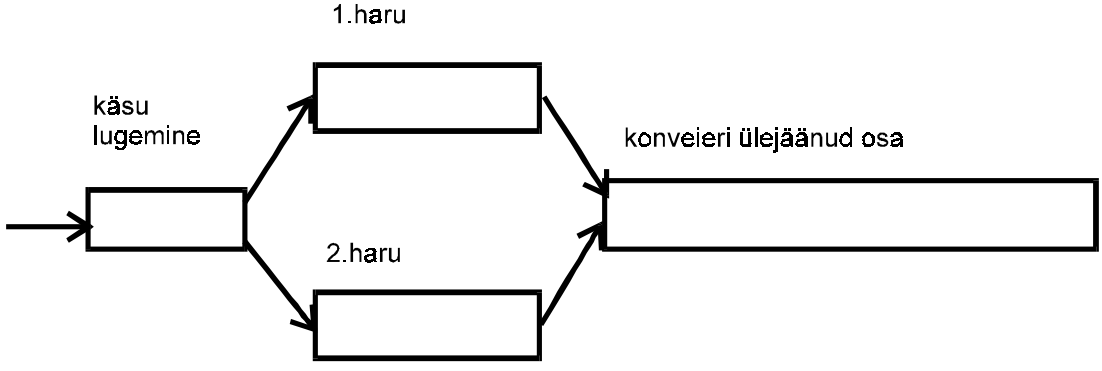
Konveieri töö on kõige efektiivsem siis, kui täidetakse üksteisele järgnevaid võimalikult ühesuguse pikkusega käske. Tingimusliku hargnemiskäsu puhul pole teada, milline kahest võimalikust käsust järgmisena täidetakse, sest see sõltub

parajasti täidetava käsu tulemusest. Kui hargnemiskäsu tulemusena ei täideta enam järgmist käsku, siis tuleb kogu konveier tühjendada ja uuesti initsialiseerida (s.t. uus esimene käsk sisse lugeda). Mida pikem on konveier, seda rohkem kaotatakse aega. Ajakulu vähendamiseks hargnemiskäskude puhul kasutatakse põhiliselt kolme erinevat strateegiat:

1. käsupuhvrid;
2. loogika kõige tõenäolisemalt täidetava käsu lugemiseks;
3. viitega hargnemiskäsud.

2.2.1. Käsupuhvrid

Käsupuhvreid kasutatakse selleks, et lugeda mälust mõlemad võimalikud täidetavad käsud. Käsupuhvrid moodustavad osa konveierist ja suurendavad konveieri pikkust.



Probleem tekib siis, kui programmis on järjestikku mitu hargnemiskäsku. Kahe järjestikulise hargnemiskäsu puhul on siis juba neli erinevat haru, 3 puhul 8, jne.

2.2.2. Kõige tõenäolisemalt täidetava käsu lugemine

Kasutatakse spetsiaalset loogikat, et ennustada, milline hargnemiskäsu harudest kõige tõenäolisemalt täidetakse. Selleks on mitmeid erinevaid meetodeid. Kõige lihtsam ja sagedamini kasutatav on strateegia kus sama hargnemiskäsu järjekordsel täitmisel eeldatakse, et täidetakse sama haru, mis eelmine kordki.

2.2.3. Viitega hargnemiskäsud

Hargnemiskäsku ei täideta mitte kohe, vaid paari-kolme hargnemiskäsule järgneva käsu järel. Siis ei ole vaja konveierit vahepeal tühjendada. Olgu näiteks kahest seadmest koosnev konveier (eraldi seadmed käsu lugemiseks ja käsu täitmiseks):

aeg	käsu lugemine	käsu täitmine
t1	hargnemiskäsu lugemine	
t2	järgmise käsu lugemine	
t3	hargnemine	järgmise käsu täitmine

2.3. Andmetevaheline sõltuvus

Olgu vaja arvutada väärtus C=2*(A+100nda mälupesa aadress) ja selle väärtuse leidmiseks järgmine käskude jada:

```
ADD   AX, [100]
SAL   AX,1
MOV   CX, AX
```

Olgu konveier 5osaline.

käsu lugemine	dekodeerimine	operandide lugemine	käsu täitmine	operandide salvestamine
ADD				
SAL	dec			
	dec	AX, [100]		
MOV			ADD	
	dec			AX
		AX		
			SHIFT	
				AX
		AX		
				CX

Väärtust registris AX ei saa korrutada kahega enne, kui liitmiskäsk on tulemuse sinna salvestanud. Samuti ei saa registri AX sisu salvestada registrisse CX enne, kui sinna on salvestatud käsu SAL tulemus.

2.4. Multistreaming

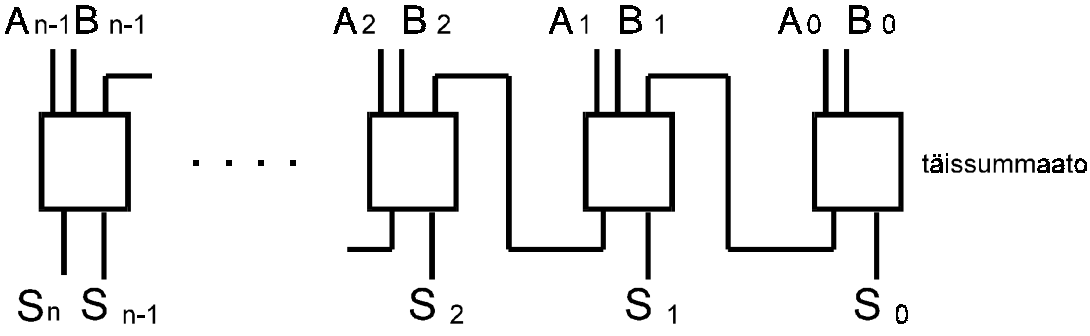
Suurema jõudlusega arvutid täidavad tihti mitut erinevat programmi, lülitudes mingite ajaintervallide järel ühe protsessi täitmiselt teisele. Sellisel juhul vähenevad tavaliselt konfliktid konveieritel, kuna ühe ja sama programmi käsud ei järgne üksteisele.

2.5. Andmekonveierid

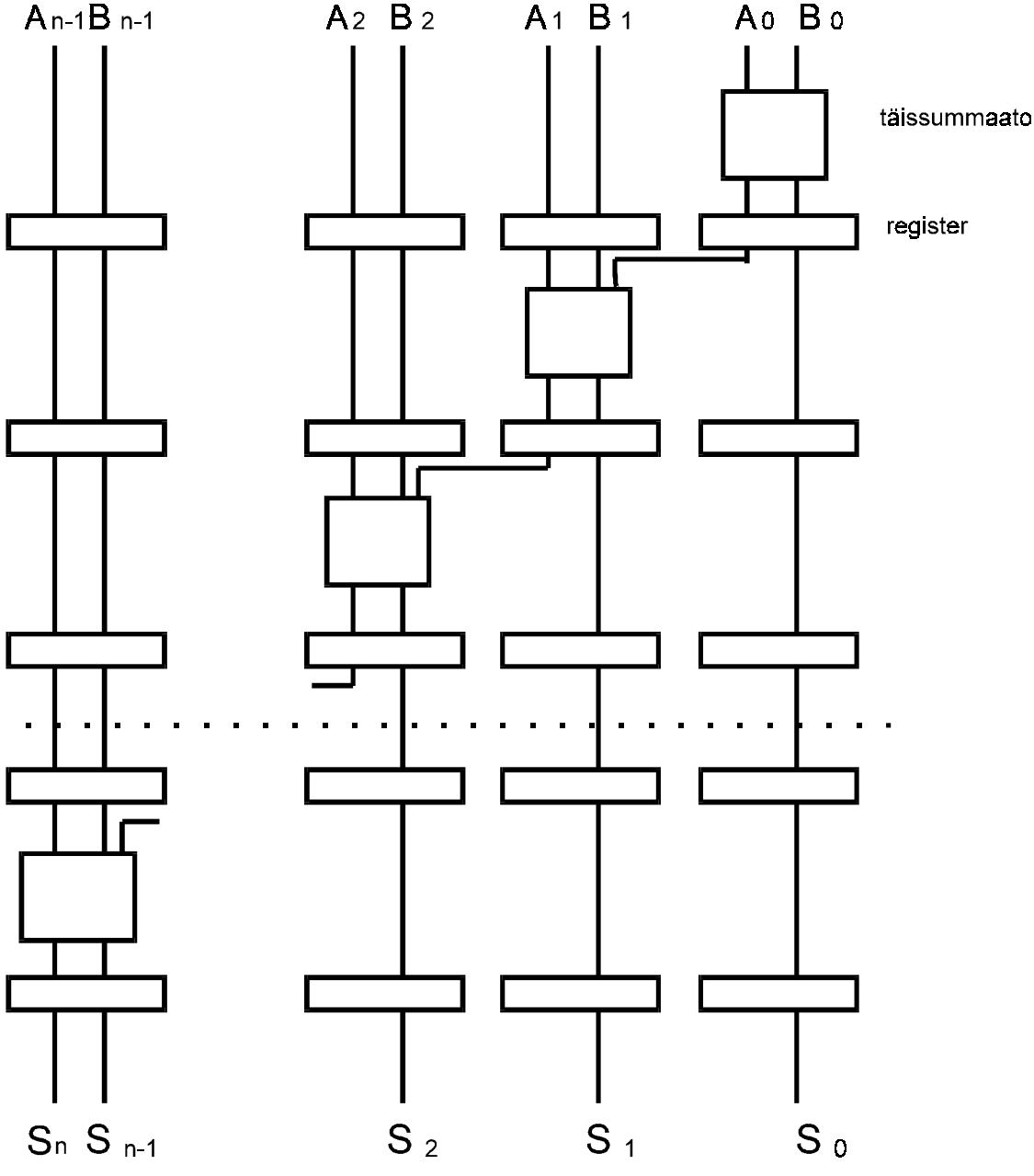
Käsukonveieri käsu täitmise osa võib endast omakorda kujutada konveierit - andmekonveierit. Andmekonveierit kasutatakse aritmeetiliste operatsioonide täitmiseks (*arithmetic processing pipeline*). Nagu käsukonveieri, nii ka andmekonveieri puhul omab konveier mõtet eelkõige siis, kui täidetakse järjestikku mitu samasugust operatsiooni (näiteks liitmistehet). Kõige sagedamini kasutatakse andmekonveiereid vektorarvutites, s.t. arvutites, mis täidavad operatsioone vektorite ja maatriksitega. Sel juhul korratakse ühte ja sama aritmeetilist operatsiooni iga vektori elemendi puhul. Skalaarprotsessorites pole andmekonveierite kasutegur nii suur. Kui andmekonveiereid skalaarprotsessorites siiski kasutatakse, on nad üldjuhul multifunktsionaalsed, s.t neid saab rekonfigureerida erinevate aritmeetikaoperatsioonide täitmiseks.

Et liita omavahel kaks *n-1*bitist arvu on vaja *n*st täissummaatorist koosnevat skeemi: Iga täissummaator liidab kaks bitti ja ülekande nooremast järgust:

Summaator:



Andmekonveier summeerimiseks:



3. Multiprotsessorsüsteemid

3.1. Flynni klassifikatsioon

- SISD (*single instruction - single data*) on kõige tavalisem ühe protsessoriga arvuti (von Neumanni arvuti).
- MIMD (*multiple instruction - multiple data*) on kõige üldotstarbelisem multiprotsessorsüsteem, kus kõik protsessorid (protsessorelemendid) töötavad samaaegselt erinevate andmetega ja täidavad erinevaid käske
- SIMD (*single instruction - multiple data*) on multiprotsessorsüsteem, kus kõik protsessorid täidavad samu käske kuid erinevate andmetega.

3.2. SIMD arvutid

SIMD arvutit kutsutatakse ka vektorarvutiks, sest sama operatsiooni täidetakse erinevate andmetega (andmete vektoriga). SIMD arvutil on üks programmimälu, kus hoitakse kõikide protsessorite poolt korraga täidetavaid käske. Samuti on tal üks keskne juhtseade, mis tagab programmi järjestikulise täitmise. Andmete jaoks võib igal protsessoril olla lokaalne mälu, kuid protsessoritel võib olla ligipääs ka ühisele globaalmälule. Kõik protsessorelemendid (PE) töötavad sünkroonselt, nad alustavad ja lõpetavad käsu täitmise samal ajal. Igal PEl on tavaliselt mõned registrid ja aritmeetika-loogikaseade. Hargnemiskäsud täidetakse juhtseadme poolt.

3.3. MIMD arvutid

MIMD arvuti puhul protsessorid täidavad samaaegselt erinevaid käske erinevate andmetega. Seetõttu on igal protsessorelemendil oma programmimälu või

juurdepääs ühisele programmimälule. Samuti on tal ka lokaalmälu andmetele või juurdepääs ühisele andmemälule. Kuna kõik PEd töötavad koos mingi ühise ülesande lahendamiseks, on vaja mehanismi informatsiooni vahetamiseks PEde vahel. MIMD multiprotsessorsüsteemid saab jaotada kahte suurde rühma sõltuvalt sellest, kuidas toimub info vahetus PEde vahel:

1. ühismäluga multiprotsessorsüsteemid;
2. ilma ühismäluta multiprotsessorsüsteemid (*message-passing systems*).

Esimesed neid kasutavad ühist globaalset mälu, mis teeb võimalikuks kõikide PEde ligipääsu kõikidele andmetele. Teist liiki multiprotsessorsüsteemide puhul on protsessorid üksteisega ühendatud ja saavad omavahel infot vahetada.

4. Paralleelsed protsessid

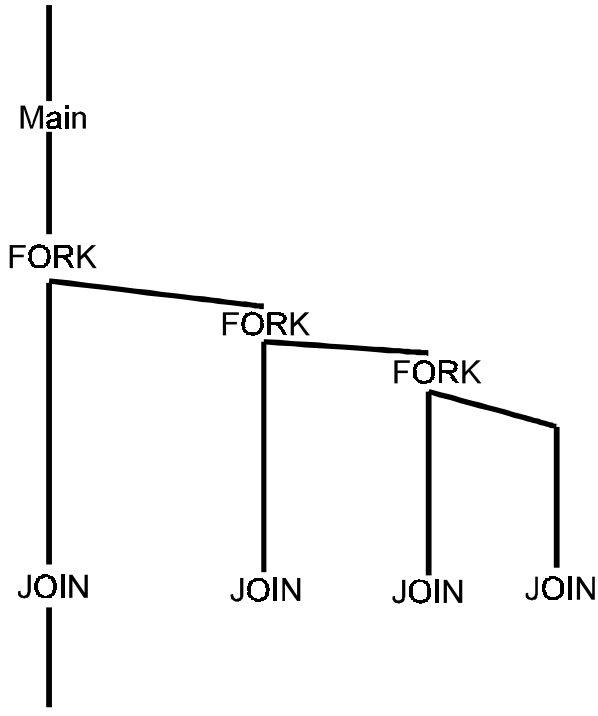
Protsessi all mõistetakse tavaliselt ühte käsku või käskude jada. Protsessil on sisendid ja väljundid (tulemused). Paralleelsete protsesside puhul räägitakse kahte liiki paralleelsusest:

1. ilmutatud paralleelsus (*explicit parallelism*). Siin määrab programmeerija ise kindlaks, millised programmi osad täidetakse paralleelselt ja millised järjestikku.
2. ilmutamata paralleelsus (*implicit parallelism*). Kompilaator määrab ära, millised programmi osad on võimalik täita paralleelselt ja koostab sellele vastavalt algoritmi.

4.1. Ilmutatud paralleelsus

Programmeerija poolt defineeritud parallelismi toetavad spetsiaalsed programmeerimiskeeled:

- Hargnemised defineeritakse käsuga `FORK` ja ühinemised käsuga `JOIN`.


Main
FORK
FORK
FORK
JOIN
JOIN
JOIN
JOIN

- Kasutatakse konstruktsiooni `PARBEGIN-PAREND`, et määrata ära laused, mis täidetakse paralleelselt:

```
PARBEGIN
   S1;
   S2;
   .
   .
   S3;
PAREND;
```

Iga lause võib omakorda olla järjestikuliselt täidetavate lausete jada:

```
PARBEGIN
   BEGIN
     .
     .
    END;
    BEGIN
     .
     .
    END;
    BEGIN
     .
     .
    END;
     .
     .
PAREND;
```

- Kasutatakse lauset `PARFOR`, et genereerida mingi kindel arv ühesuguseid paralleelselt täidetavaid protsesse:

```
PARFOR I:=1 TO n DO
```

```
BEGIN
  S1;
  S2;
  .
  .
  Sm
END;
```

See programm genereerib *n* protsessi, igaüks nendest sisaldab lauseid *S1, S2 ...Sn.*

4.2. Ilmutamata paralleelsus

4.2.1. Bernsteini tingimused

Bernsteini tingimused määravad kindlaks, kas kahte protsessi on võimalik täita paralleelselt.

I_i on mälupesad hulk, millede poole pöördub lugemiseks protsess P_i.

O_i on mälupesad hulk, millede poole pöördub kirjutamiseks protsess P_i.

Protsesse P_1 ja P_2 saab täita paralleelselt kui

$I_1 \cap O_2 = \varnothing$
$I_2 \cap O_1 = \varnothing$
$O_1 \cap O_2 = \varnothing$

I_i on omistamislause vasakpoolne muutuja ja O_i parempoolne:

```
A := X+Y;
B := X+Z;
```

Neid kahte lauset saab täita paralleelselt, sest

$I_1 = (X, Y)$
$I_2 = (X, Z)$
$O_1 = (A)$
$O_2 = (B)$

ja seega Bersteini tingimused on täietud.

Bersteini tingimused kehtivad ka enama kui kahe protsessi puhul. Siis

$$\forall i, j(i \neq j)\begin{cases} I_i \cap O_j = \varnothing \\ I_j \cap O_i = \varnothing \\ O_i \cap O_j = \varnothing \end{cases}$$

Kas neid lauseid saab täita paralleelselt?

```
A:=X+Y;
B:=X*Y;
C:=Y-X;
```

Parallelismi esineb tihti kõrgkeelte tsüklilistes lausetes. Näiteks

```
FOR I:=1 TO 20 DO
      A[i]:=B[i];
```

Selline tsükkel koosneb tegelikult 20st lausest:

```
A[1]:=B[1];
A[2]:=B[2];
A[3]:=B[3];
      .
      .
A[19]:=B[19];
A[20]:=B[20];
```

Kõiki neid lauseid saab täita paralleelselt, sest Bernsteini tingimused on täidetud.

4.3. Kriitiline intervall

Et mitu paralleelset protsessi saaks korraga olla aktiivsed on vaja mehanismi nende töö sünkroniseerimiseks. Ka teineteisest sõltumatud protsessid pöörduvad aeg-ajalt ühiste ressursside poole (näiteks mälu). Et ainult üks

protsess saaks korraga selliste ressursside poole pöörduda, jagatakse kõik protsesid kriitilisteks intervallideks. Kriitilises intervallis saab olla korraga ainult üks protsess, s.t. kui üks protsess mingit ressurssi kasutab, on teistel keelatud seda teha.

4.4. Lukud (locks)

Lukud on kõige lihtsam viis kriitiliste intervallide vastastikuseks välistamiseks. Lukk on 1bitine muutuja. Kui tema väärtus on 1 siis see näitab, et protsess on sisenenud kriitilisse intervalli (lukk on kinni). Kui luku väärtus on 1, siis pole ühtegi protsessi parajasti kriitlises intervallis (lukk on lahti). Kui protsess soovib siseneda kriitilisse intervalli peab ta kõigepealt kontrollima, millises seisus on lukk. Kui lukk onl lahti, siseneb protsess kriitilisse intervalli ja suleb luku. Kriitilisest intervallist väljudes teeb ta luku uuesti lahti. Kui protsess tahab siseneda kriitilisse intervalli, aga lukk on kinni, siis ta peab ootama. Kõige lihtsamal juhul ta lihtsalt kontrollib pidevalt, kas lukk on juba lahti (*busy waiting*), tehes samal ajal mitte midagi muud kasulikku. Kui rohkem kui üks protsess ootab luku avanemist, võib olla vaja mehanismi, mis avanemisel annaks kriitilisse intervalli sisenemise õiguse kõige suurema prioriteediga protsessile. On oluline, et protsessi ei saaks katkestada luku kontrollimise ja intervalli sisenemise vahepeal, s.t, et ei tekiks olukorda, kus protsess on avastanud, et lukk on lahti, kuid kriitilisse intervalli sisenemisel leiab, et teine protsess on temast ette jõudnud. Paljudel protsessoritel on luku avamiseks ja lukustamiseks nähtamatud masinkäsud, millede täitmine on realiseeritud riistvaraliselt.

4.4.1. LOCK

`LOCK` käsu puhul genereeritakse `LOCK` signaal selle ja järgmise käsu (luku lukustamise) ajaks. `LOCK` signaal keelab riistvaraliselt selle signaali ajal täidetava käsu katkestamise.

4.4.2. Test And Set(TAS)

Käsul Test And Set on kuju

```
TAS mälu aaderss
```

TAS kontrollib, milline väärtus on selle aadressil ja seejärel muudab väärtuse 1ks, s.t suleb luku vaatamata sellele, kas ta oli juba enne suletud või mitte.

4.5. Semafoorid

Semafoor *s* võib omada kas positiivset täisarvulist väärtust või olla null. Semafooril on defineeritud kaks operatsiooni - P ja V:

P(S): kui s=0, siis protsessi täitmine lükatakse edasi seniks, kui s on positiivne. Kui s on positiivne arv, siis tema väärtust vähendatakse ühe võrra ja protsess võib jätkata.

V(S): suurendab semafoori väärtust 1 võrra.

Kahendsemafoori puhul võib s omada ainult väärtusi 1 ja 0. Kui s=1, siis pole ühtegi protsessi kriitilises intervallis. Kui siis mõni protsess soovib siseneda kriitilisse intervalli, omistab ta semafoorile väärtuse s=0. Siis ei saa enam ükski protsess kriitilisse intervalli siseneda, kuid kõik protsessid, mis seda teha sooviksid, pannakse järjekorda ootama. Kui protsess väljub kriitilisest intervallist rakendab ta V(s) operaatorit, lubades sellega siseneda järgmisel protsessil:

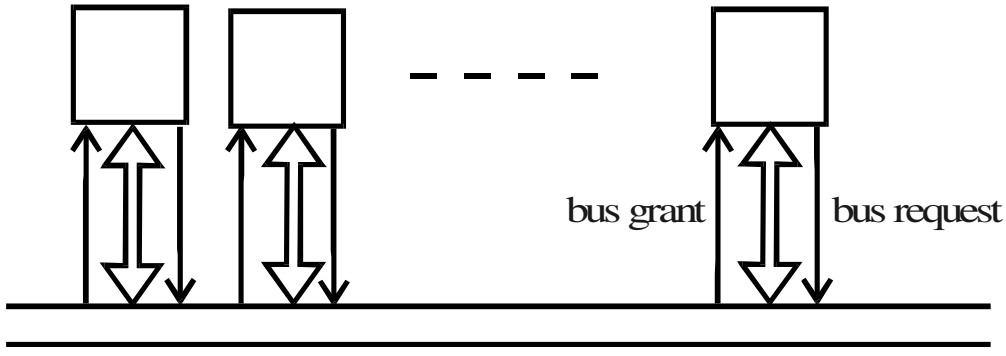
```
P(s)
Kriitiline intervall
V(s)
```

Semafoorid aitavad vältida surnud seisu. Surnud seis tekib siis, kui kõik protsessid tahavad kasutada mõnda ressurssi, kuid see on parajasti hõivatud mõne teise poolt ja ükski protsess ei saa jätkata. Nälgimine tekib siis, kui üks või osa protsesse ei saa jätkata, sest teised protsessid hoiavad vajaminevaid resursse kinni.

4.6. Monitorid

Monitoride kasutamise puhul saab ressursi kasutamine toimuda ainult monitori kaudu. Igal ajahetkel võib ainult üks protsess monitori kasutada. Kui protsess tahab kasutada monitori, mis on parasjagu kasutusel teise protsessi poolt, pannakse ta järjekorda ootama kuni monitor vabaneb. Monitori vabanemisel saab seda kasutada esimene protsess järjekorrast.

5. Ühissiiniga multiprotsessorsüsteemid

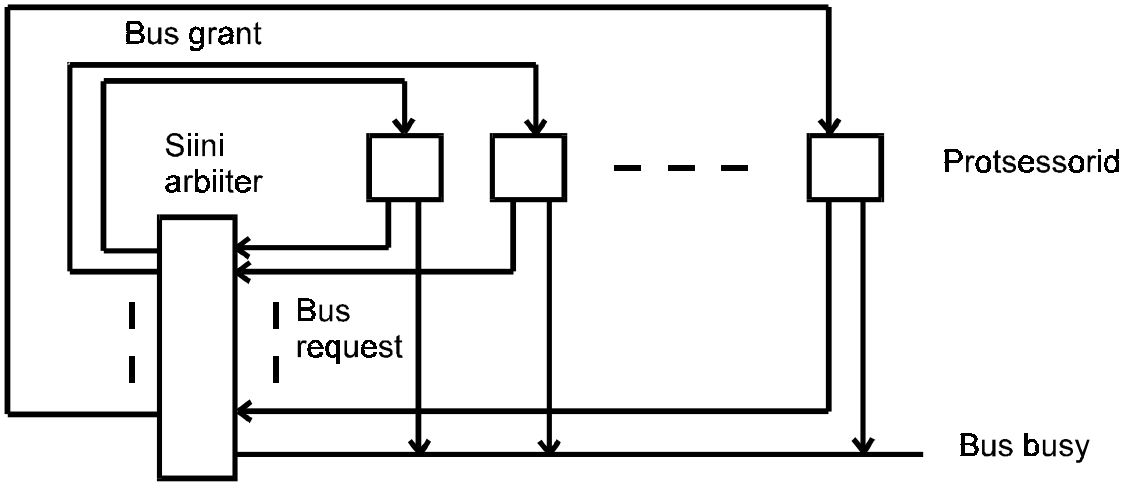


Ühise siini külge on ühendatud nii protsessorid kui mälumoodulid. Mälumoodulid kokku moodustavad kogu süsteemi ühise mälu, millele on ligipääs kõikidel protsessoritel. Siini saab kasutada korraga ainult üks protsessor. Protsessorit, millel on parasjagu siini kasutamise õigus, nimetatakse siini haldajaks (*bus master*). Kui protsessor soovib siini haldamise õigust, saadab ta päringu (*bus request*) siini haldajale. Kui haldaja on siini kasutamise lõpetanud saadab ta kasutamiset taotlevale protsessorile kinnitussignaali (*bus grant*) ja kasutamist taotlev protsessor saab uueks siini haldajaks. Tihti kasutatakse ka signaali *bus busy*, millega siini haldur võib vastata, kui ta pole siini kasutamist veel lõpetanud.

Kui siini soovib kasutada korraga mitu protsessorit, on vaja mehanismi konfliktide lahendamiseks (siini arbitreerimiseks). Arbitreerimine jaotatakse paralleelseks ja järjestikuliseks ning ka tsentraliseerituks ja detsentraliseerituks. Protsessorite prioriteedid võivad olla kas staatilised või dünaamilised. Staatiliste prioriteetide puhul on igal protsessoril alati üks ja sama prioriteet. Dünaamiliste prioriteetide puhul nende järjestust muudetakse, et anda kõigile protsessoritele

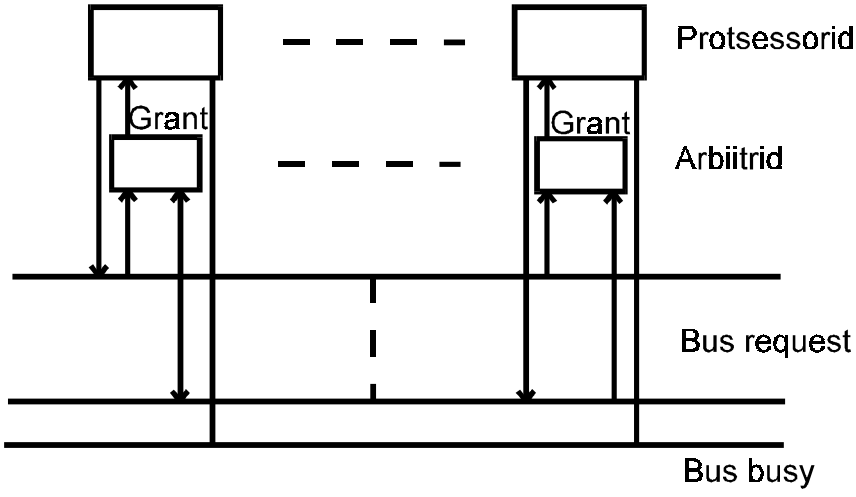
siini kasutamise võimalus. Sünkroonsete süsteemide puhul võib siini kasutamine olla jaotatud võrdselt kõigi protsessorite vahel nii, et igal protsessoril on oma kindel ajaintervall siini kasutamiseks.

5.1. Paralleelne arbitreerimine



Bus request signaalid saadetakse siini arbiiterile. Siini arbiiter otsustab, milline protsessor saab siini kasutamise õiguse. Signaal *bus busy* aktiveeritakse protsessori poolt, mis parasjagu siini kasutab. Signaalid *bus busy* ja *bus request* on aktiivsed kogu aja kui protsessor siini kasutab. Signaal *bus grant* on aktiivne kogu aja kui protsessoril on siini kasutamise õigus. Kui mingi protsessor kasutab siini, kuid samal ajal tahab seda kasutada mõni teine, kõrgema prioriteediga protsessor, annab arbiiter *bus grant* signaali kõrgema prioriteediga protsessorile. Tavaliselt ei ole siini haldur võimeline siini koheselt vabastama (näiteks on pooleli kirjutamis/lugemisoperatsioon). Kui siini haldur märkab, et ta on kaotanud *bus grant* signaali, vabastab ta siini esimesel võimalusel. *Bus busy*

signaali hoiab ta aktiivsena kogu siini kasutamise aja. Kõrgema prioriteediga protsessor saab siini haldamise üle võtta siis, kui tal on *bus grant* signaal ja *bus busy* ei ole enam aktiivne. Selline siini arbitreerimine on tsentraliseeritud. Detsentraliseeritud arbitreerimise puhul on igal protsessoril oma siini arbiiter.



Bus request signaalid jõuavad ühise siini kaudu kõikide arbiitriteni, kuid iga arbiiter genereerib ainult ühe *bus grant* signaali. Selline detsentraliseeritud arbitreerimine on mõnevõrra töökindlam.

5.1.1. Dünaamilised prioriteedid

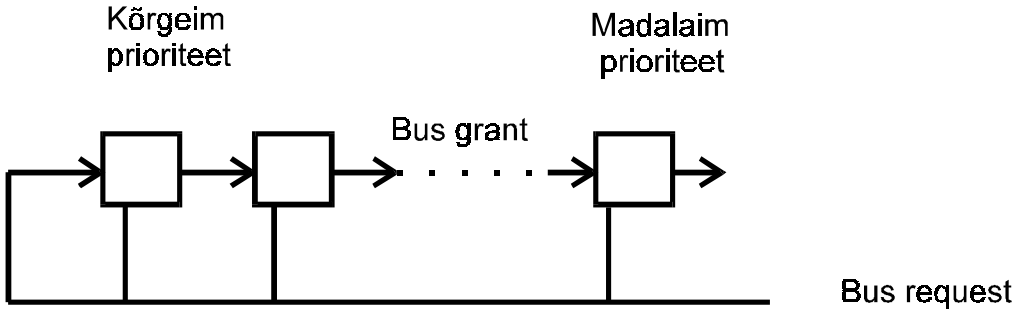
Nii staatiliste kui ka dünaamiliste prioriteetide puhul määratakse prioriteedid kindlaks siini arbiitris. Dünaamiliste prioriteetide eesmärk on anda kõikidele protsessoritele võrdne ligipääs siinile. Prioriteetide muutmiseks kasutatakse mitmesuguseid erinevaid algoritme:

- Kõikide protsessorite priroteete vähendatakse ühe võrra ja kõige madalama prioriteediga protsessor saab kõige kõrgema prioriteedi.
- Protsessorile, mis just lõpetas siini kasutamise, antakse kõige madalam prioriteet ja kõik ülejäänud protsessorid saavad ühe koha võrra kõrgema prioriteedi.
- Prioriteedid genereeritakse juhuslikult.

- Kõige kõrgem prioriteet antakse protsessorile, mis pole siini kõige pikemat aega kasutanud.

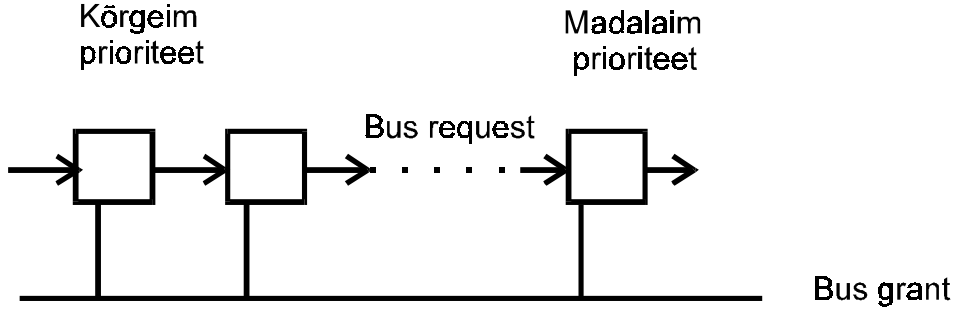
5.2. Järjestikuline arbitreerimine

Järjestikulise arbitreerimise puhul kasutatakse signaali, mis järjestikku läbib kõiki ühissiiniga ühendatud protsessoreid. See signaal võib olla kas *bus grant* või *request.* Kõige sagedamini kasutatakse *bus grant* signaali.



Kui üks või enam protsessorit soovivad siini kasutada, väljastavad nad *request* signaali. Signaal läbib kõik protsessorid alates kõige kõrgema prioriteediga protsessorist ja lõpetades kõige madalamaga. Esimene protsessoritest, mis väljastas *request* signaali takistab seda edasi levimast.

Kui kasutatakse *bus request* signaali, siis see läbib samuti järjestikku kõik protsessorid, alustades sellest, mis on kõige kõrgema prioriteediga.



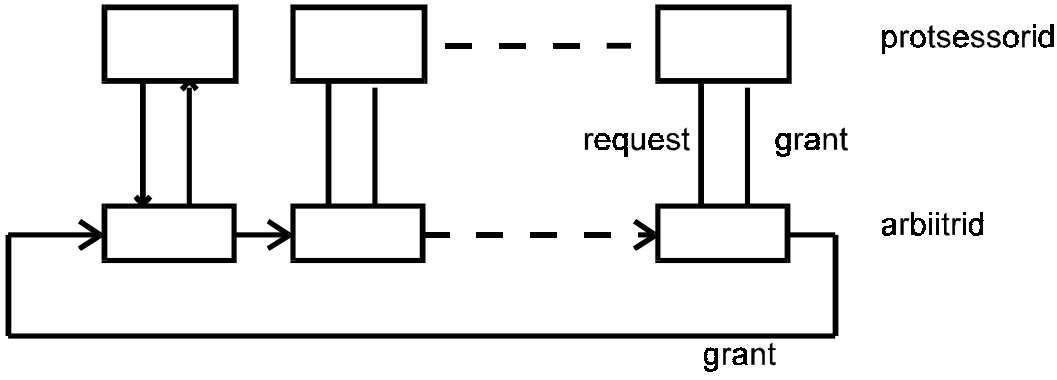
Iga prtosessor, mis soovib siini haldamise õigust, genereerib *request* signaali. See liigub vasakult paremale madalama prioriteediga protsessorite suunas, kuni jõuab siini haldurini. Kuna signaal saabus siini haldurisse vasakult, s.t. kõrgema prioriteediga protsessorite poolt, vabastab siini haldur siini ja genereerib *bus*

grant signaali. Seda signaali märkavad kõik siini haldamist taotlevad protsessorid, kuid ainult ühel neist puudub sisendis *request* signaal. See protsessor saab siini haldamise õiguse.

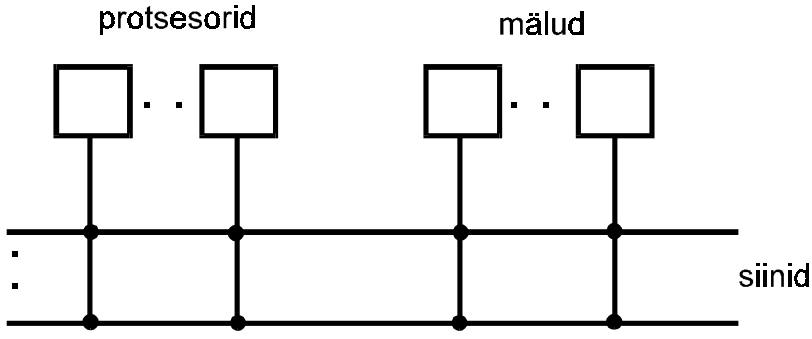
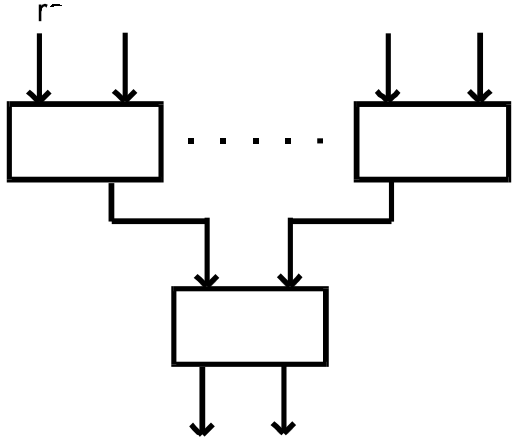
Mitte kummalgi järjestikulise arbitreerimise juhul ei tohi uus siini haldur siini enne üle võtta, kui ta on vaba. Selle garanteerimiseks on kaks võimalust:

1. protsessorid ei tohi *request* signaali väljastada enne, kui siin on vaba.
2. *request* signaali võib väljastada igal ajal, aga siini ei tohi üle võtta enne, kui ta on vaba.

5.2.1. Dünaamilised prioriteedid



Siini haldajal on kõige madalam prioiteet, temast paremal asuval protsessoril kõige kõrgem ja nii ringiratast kahanevas järjekorras. Siini haldaja väljastab *bus grant* signaali. Kui mõnel protsessoril on *bus request* signaal aktiivne, takistab ta *bus grant* signaali levimise. Kui siini haldur märkab, et signaal enam temasse tagasi ei jõua, vabastab ta siini nii ruttu kui võimalik.

6. Interconnection network

6.1. Mitme ühissiiniga multiprotsessorsüsteemid