FKEF.02.143

Arvuti arhitektuur

Computer Architecture and Organization

dotsent
Toomas Plank

FKEF.02.143

Vahemälu

Cache

11. loeng,
9. mai 2008

Vajadus kiire mälu järele

- Võrreldes protsessori kiirusega on mälu aeglane seade
- Vahemälu võimaldab mälul näida kiiremana kui ta tegelikult on
- Vahemälu efektiivsus põhineb järgmistel tõdemusel:
  - Enamus programmide tööajast kulub tsüklites ja samade protseduuride sagedases täitmises
  - Hiljuti täidetud käsud tulevad suure tõenäosusega varsti uuesti täitmisele
  - Viimati täidetud käsule lähedal (mälus paiknemise mõttes) paiknevad käsud tulevad samuti suure tõenäosusega peagi täitmisele

Vahemälu (*Cache*)

- Paigutades programmi aktiivse osa kiiresse vahemälusse, saame suure võidu programmi tööajas
- Eelmisel slaidil kirjapandud **ajalise läheduse tõdemus** soovitab lugeda kõik vajaminevad käsud vahemällu
  - tõenäoliselt läheb neid varsti uuesti tarvis
- Eelmisel slaidil kirjapandud **ruumilise läheduse tõdemus** soovitab lugeda vahemällu mitte üksikud käsud vaid ka nende naabruses asuvad käsud
  - s.t loeme andmed vahemällu plokkide kaupa (*block, cache line*)
  - tõenäoliselt läheb ka neid käske varsti tarvis

Põhimälu ja vahemälu

- Kui vajatakse mälust andmeid, siis loetakse need vahemällu
- Kui vajalikud andmed on juba vahemälus olemas, siis loetaksegi kohe sealt
- Tagatakse vastavus põhimälu ja vahemälus resideeruva koopia vahel (*mapping function*)
- Kui vahemälu on täis, tuleb uute andmete lisandumisel miskit olemasolevat vahemälust välja visata (*replacement algorithm*)

Vahemälu töökorraldus

- Protsessor ei pea olema teadlik vahemälu olemasolust,
  - tema annab ainult mälust lugemise ja mälusse kirjutamise käske
  - Vahemälu kontrollahelad selgitavad välja, kas vajalik info on juba vahemälus (tabamus)
  - või tuleb info alles vahemälusse lugeda (möödalask)
- Kui andmed on juba vahemälus, siis põhimälust lugeda pole tarvis

Kirjutamiseks on kaks varianti:

- *write-through* – nii vahemälu kui ka mälu uuendatakse üheaegselt
- *write-back, copy-back* – uuendatakse ainult vahemälu
  - Vahemälust ploki eemaldamisel uuendatakse info lõpuks ka põhimälus
  - Info uuendamise vajaduse kohta kajastub muudetud lipubitis (*modified, dirty bit*)

Vahemälu töökorraldus

- Lugemine. Kui infot pole vahemälus, siis
  - loetakse kogu plokk põhimälust vahemällu ja seejärel edastatakse vajalik sõna protsessorile
  - loetakse info põhimälust ja edastatakse vajalik sõna protsessorile kohe kui ta on olemas (veidi kiirem, aga nõuab keerukamaid juhtimisahelaid)
- Kirjutamine. Kui infot pole vahemälus, siis
  - Kirjutatakse info põhimällu ilma vahemälu vahenduseta (*write-through*)
  - Loetakse esmalt kogu plokk vahemällu ja siis muudetakse vastava sõna vahemälus (*write-back*)

Info paigutus vahemälus

Paigutusviisid

- Otsene paigutus (*Direct Mapping*)
- Assotsiatiivne paigutus (*Associative Mapping*)
- Assotsiatiivse rühmana paigutus (*Set-Associative Mapping*)

Vaatleme neid paigutusviise lähemalt, kasutades näitena

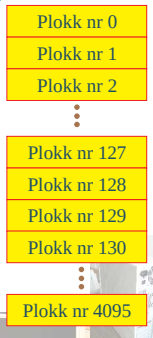
- 4096 plokist koosnevat põhimälu
- 128 plokist koosnevat vahemälu
- Igas plokis olgu meil 16 sõna
- Seega põhimälu 16×4096 = 65536 sõna
- ja vahemälu 16×128 = 2048 sõna

Otsene paigutus (*Direct Mapping*)

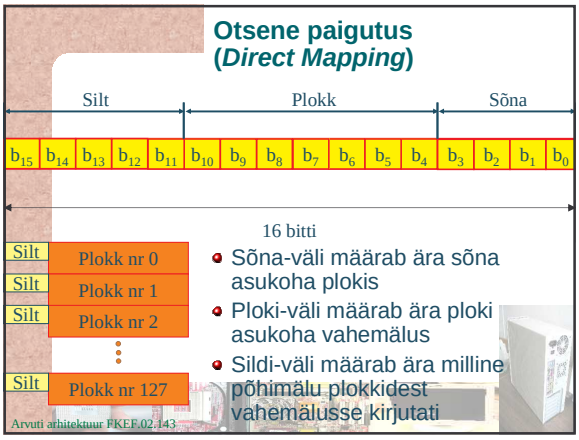
- Mälu plokk *j* pannakse *C* plokist koosnevas vahemälus plokki

$$j \bmod C$$

- Mälu aadress jagatakse kolmeks osaks
- Käesolevas näites on meil 16-bitised mälu aadressid



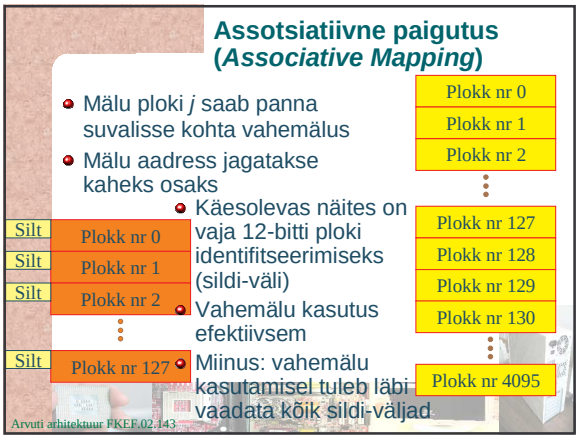
Otsene paigutus (*Direct Mapping*)



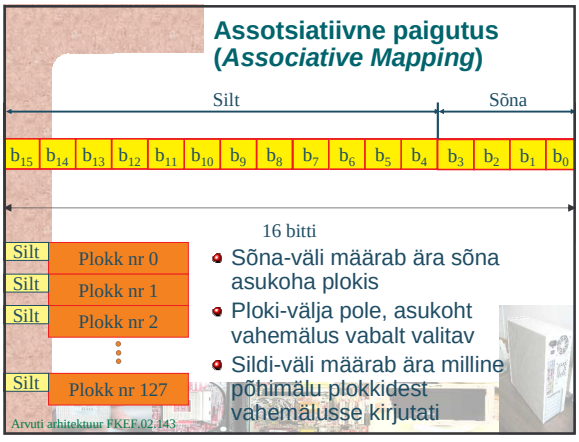
- Sõna-väli määrab ära sõna asukoha plokis
- Ploki-väli määrab ära ploki asukoha vahemälus
- Sildi-väli määrab ära milline põhimälu plokkidest vahemälusse kirjutati

Assotsiatiivne paigutus (*Associative Mapping*)

- Mälu ploki *j* saab panna suvalisse kohta vahemälus
- Mälu aadress jagatakse kaheks osaks
  - Käesolevas näites on vaja 12-bitti ploki identifitseerimiseks (sildi-väli)
  - Vahemälu kasutus efektiivsem
  - Miinus: vahemälu kasutamisel tuleb läbi vaadata kõik sildi-väljad



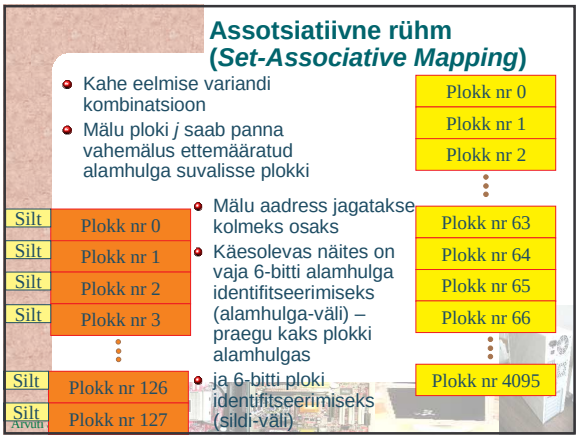
Assotsiatiivne paigutus (*Associative Mapping*)



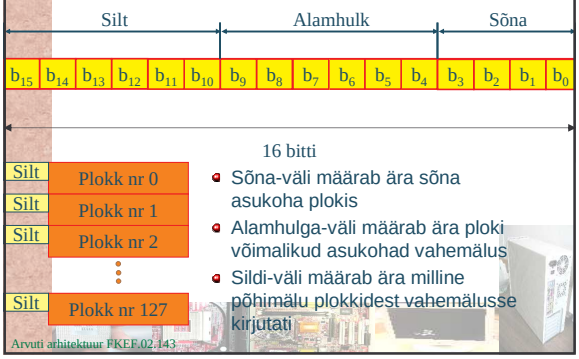
- Sõna-väli määrab ära sõna asukoha plokis
- Ploki-välja pole, asukoht vahemälus vabalt valitav
- Sildi-väli määrab ära milline põhimälu plokkidest vahemälusse kirjutati

Assotsiatiivne rühm (*Set-Associative Mapping*)

- Kahe eelmise variandi kombinatsioon
- Mälu ploki *j* saab panna vahemälus ettemääratud alamhulga suvalisse plokki
- Mälu aadress jagatakse kolmeks osaks
- Käesolevas näites on vaja 6-bitti alamhulga identifitseerimiseks (alamhulga-väli) – praegu kaks plokki alamhulgas
- ja 6-bitti ploki identifitseerimiseks (sildi-väli)



Assotsiatiivne rühm (*Set-Associative Mapping*)



- Sõna-väli määrab ära sõna asukoha plokis
- Alamhulga-väli määrab ära ploki võimalikud asukohad vahemälus
- Sildi-väli määrab ära milline põhimälu plokkidest vahemälusse kirjutati

Asjakohasuse bitt *Valid bit*

- See bitt ütleb, kas andmed vahemälus on asjakohased
- Arvuti sisselülitamisel kõik need bitid väärtusega **0**
- Info laekumisel seatakse selle biti väärtuseks **1**
- Kui põhimälu uuendatakse vahemälust mööda minnes, tuleb kontrollida selle ploki olemasolu/puudumist vahemälus
  - kui see plokk vahemälus, siis seatakse selle biti väärtuseks **0**
- Samalaadsed mured DMA ülekannete puhul mälust kõvakettale
  - Siin on võimalik olukord, kus põhimälus ei ole adekvaatne info (*write-back protocol*)
  - Üks võimalus andmed jõuga kaasajastada ja alles siis DMA ülekanne teha
  - Seda probleemi nimetatakse vahemälu koherentsuse küsimuseks

Asendusalgoritmid

- Otsese paigutuse korral on põhimälu ploki paigutuskoht vahemälus üheselt teada ja pole vaja pead murda selle üle, millist plokki vahemälus asendada
- Assotsiatiivse paigutuse ja assotsiatiivse rühma paigutuse korral on valik suurem
- NB! Asendusalgoritmi valik mõjutab programmi jõudlust väga palju!
- LRU (*least recently used*) *replacement algorithm* Vahetame välja ploki, mille viimatisest kasutamisest on kõige rohkem aega möödas
  - kasutusinfo jaoks peab olema spetsiaalne loendur
- Vanima elemendi väljavahetamine
  - miinus: ei arvesta hiljutisi kasutusmustreid
- Juhusliku ploki asendamine
  - Vaatamata lihtsusele üllatavalt head tulemused!

Näide

- Oletame, et mõõtsime kolme patarei pinget, igat patareid 9 korda
- Mõõdised M panime arvuti mällu 3×9 maatriksina aadressidele 2700…271A

0010 0111 0000 0000	M(0,0)
0010 0111 0000 0001	M(1,0)
0010 0111 0000 0010	M(2,0)
0010 0111 0000 0011	M(0,1)
0010 0111 0000 0100	M(1,1)
0010 0111 0000 0101	M(2,1)
...	...
0010 0111 0001 1000	M(0,8)
0010 0111 0001 1001	M(1,8)
0010 0111 0001 1010	M(2,8)

Sildi väli:

Näide: arvutusülesanne

- Oletame, et meil on vaja leida iga mõõdise hälve aritmeetilisest keskmisest ja kirjutada see mõõdise asemel samasse mälupesasse
- C# programm selle rehkenduse tegemiseks võiks olla alljärgnev:

```
for (int j = 0; j < 3; j++)
{
     //Leiame rea summa
     double Rea_Summa = 0;
     for (int i = 0; i < 9; i++)
          Rea_Summa += M[j, i];
     //Leiame keskmise
     double Keskmine = Rea_Summa / 9;
     //Rehkendame hälbed
     for (int i = 0; i < 9; i++)
          M[j, i] = M[j, i] - Keskmine;
}
```

Otsese paigutusega vahemälu

0	…
1	…
2	…
3	…
4	…
5	…
6	…
7	…

- Vahemälu pesa aadressi saame mälu aadressist tehtega mod 8
- Sisuliselt määravad asukoha ära mälu aadressi kolm viimast bitti

M(0,0)
M(0,1)
M(0,2)
M(0,3)
M(0,4)
M(0,5)
M(0,6)
M(0,7)
M(0,8)

Assotsiatiivse paigutusega vahemälu

0	…
1	…
2	…
3	…
4	…
5	…
6	…
7	…

- Esimene tsükkel
  i = 0 … 8
- Teine tsükkel
  i = 0 … 8

M(0,0)
M(0,1)
M(0,2)
M(0,3)
M(0,4)
M(0,5)
M(0,6)
M(0,7)
M(0,8)

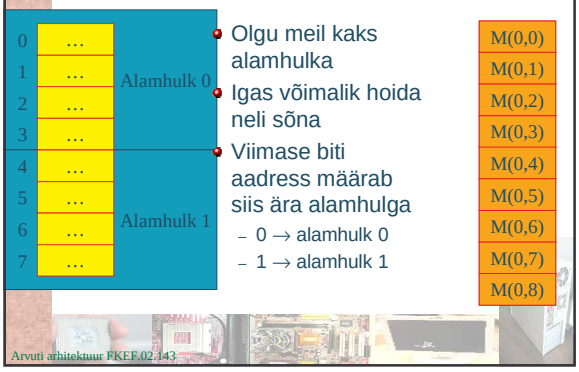
Assotsiatiivse paigutusega vahemälu

0	…
1	…
2	…
3	…
4	…
5	…
6	…
7	…

- Esimene tsükkel
  i = 0 … 8
- Teine tsükkel
  i = 8 … 0

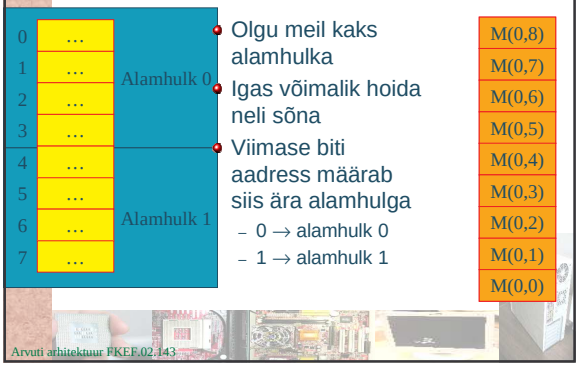
M(0,8)
M(0,7)
M(0,6)
M(0,5)
M(0,4)
M(0,3)
M(0,2)
M(0,1)
M(0,0)

Vahemälu assotsiatiivse rühmana



- Olgu meil kaks alamhulka
- Igas võimalik hoida neli sõna
- Viimase biti aadress määrab siis ära alamhulga
  - 0 → alamhulk 0
  - 1 → alamhulk 1

Vahemälu assotsiatiivse rühmana



- Olgu meil kaks alamhulka
- Igas võimalik hoida neli sõna
- Viimase biti aadress määrab siis ära alamhulga
  - 0 → alamhulk 0
  - 1 → alamhulk 1

FKEF.02.143

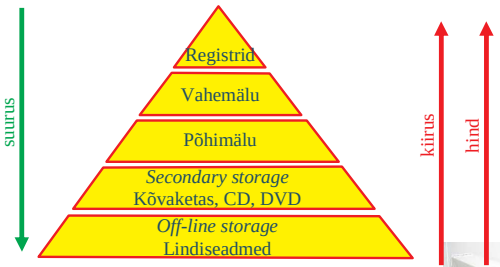
Jõudlus

Performance

Jõudlust mõjutavad tegurid

- Kaks peamist tegurit, mis mõjutavad konkreetse arvuti ärilist edukust on
  - hind
  - jõudlus
- Soovime saada parimat jõudlust võimalikult madala hinnaga
- Mälu puhul on tähtis,
  - kui kiiresti saame käsud/andmed mälust protsessorisse
  - kui suur on mälu maht

Mälu hierarhia



Tabavus (*hit rate*)

$$Tabavus = \frac{Sõna\ vahemälust\ leidmiste\ arv}{Mälupöördumiste\ koguarv}$$

- Näide:
  - Oletame, et sõnade mälust lugemisel leiti 952 juhul sõna vahemälust
  - ja 47 juhul sõna vahemälus ei olnud
- siis tabavus

$$Tabavus = \frac{952}{952+47} = \frac{952}{999} = 95{,}3\%$$

Efektiivne pöördumisaeg

$$Efektiivne\ pöördumisaeg = \frac{tabamusi \times t_{tabamus} + möödalaske \times t_{möödalask}}{Mälupöördumiste\ koguarv}$$

- Näide:
  - Oletame, et sõnade mälust lugemisel leiti 952 juhul sõna vahemälust
  - ja 47 juhul sõna vahemälus ei olnud
  - vahemälust lugemisele kulus 25 ns
  - põhimälust lugemisele koos vahemällu kirjutamisega kulus 2000 ns

$$Efektiivne\ pöördumisaeg = \frac{952 \times 25\ \text{ns} + 47 \times 2000\ \text{ns}}{952 + 47}$$

$$= \frac{23800\ \text{ns} + 94000\ \text{ns}}{999} = 118\ \text{ns}$$

Mitmetasemeline vahemälu

$$Tabavus_{L1} = \frac{Sõna\ vahemälust\ L1\ leidmiste\ arv}{Mälupöördumiste\ koguarv}$$

$$Tabavus_{L2} = \frac{Sõna\ vahemälust\ L2\ leidmiste\ arv}{Sõna\ vahemälust\ L1\ mitteleidmiste\ arv}$$

$$Efektiivne\ pöördumisaeg = \frac{L1\ tabamusi \times t_{L1} + L2\ tabamusi \times t_{L2} + L2\ möödalaske \times t_{möödalask}}{Mälupöördumiste\ koguarv}$$

Näide

- Oletame, et sõnade mälust lugemisel leiti 9 520 juhul sõna vahemälust L1
- 470 juhul sõna vahemälus L1 ei olnud
- ja 10 juhul polnud seda sõna ka vahemälus L2
- vahemälust L1 lugemisele kulus 5 ns
- vahemälust L2 lugemisele kulus 25 ns
- põhimälust lugemisele koos vahemällu kirjutamisega kulus 200 ns

- siis efektiivne pöördumisaeg

$$Efektiivne\ pöördumisaeg = \frac{9520 \times 5\ \text{ns} + 460 \times 25\ \text{ns} + 10 \times 200\ \text{ns}}{9520 + 470}$$

$$= \frac{47600\ \text{ns} + 11500\ \text{ns} + 2000\ \text{ns}}{9990} = \frac{61100\ \text{ns}}{9990} = 6{,}12\ \text{ns}$$

Kasutatud kirjandus

- Carl Hamacher, Zvonko Vranesic, Safwat Zaky, Computer organization 5th edition (2002) 805 p.
- Miles Murdocca, Vincent Heuring, Computer Architecture and Organization: An Integrated Approach (2007) 544 p.