Увод у организацију и архитектуру рачунара 2

Александар Картељ

kartelj@matf.bg.ac.rs

Напомена: садржај ових слајдова је преузет од проф. Саше Малкова

Кеш меморија

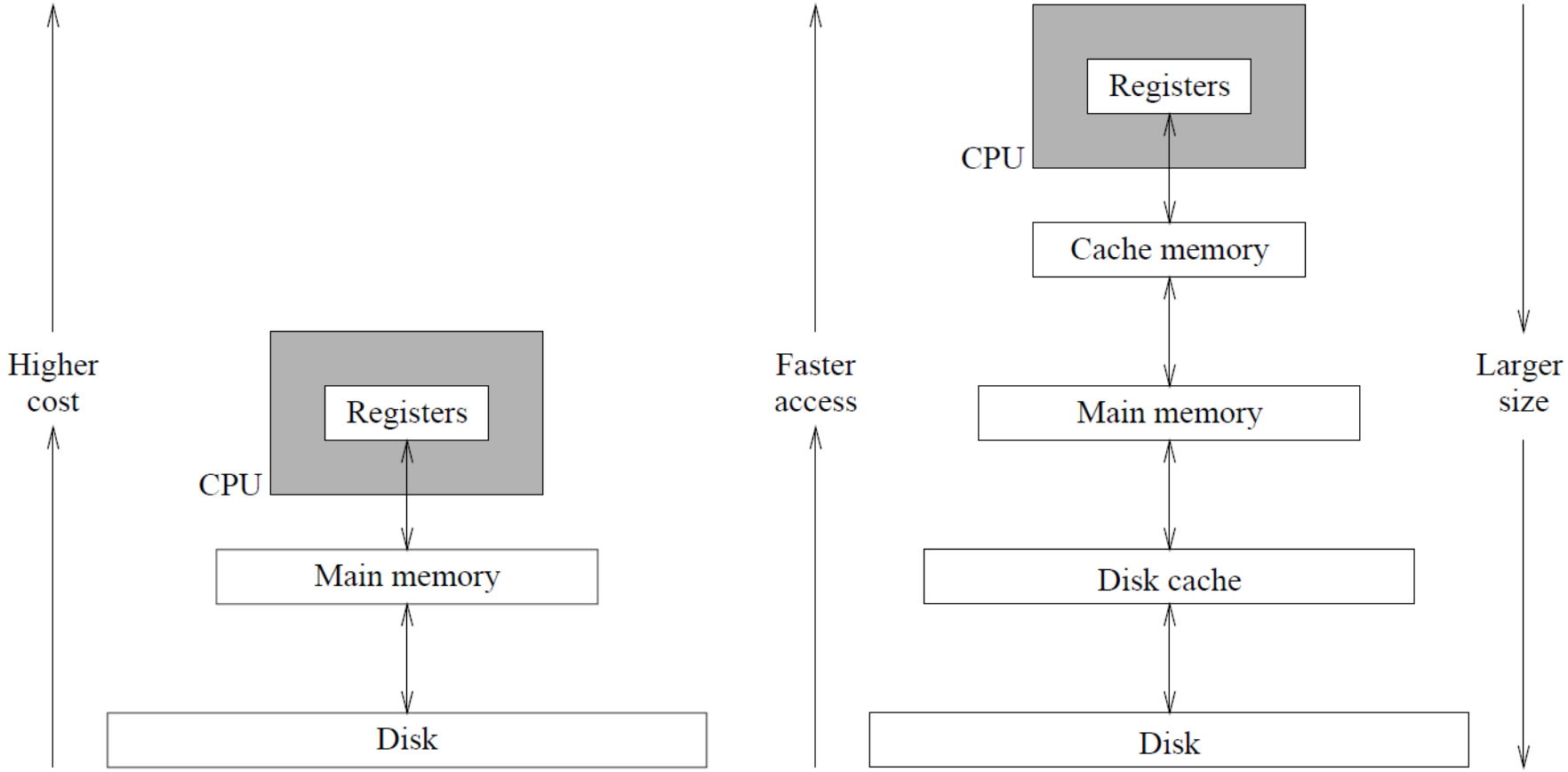
Увод

Намена кеша

- Процесор садржи регистре, као најефикаснију меморију у систему
- Радна меморија рачунара је релативно велика и спора
- Разлика у брзини ова два слоја је довољно велика
  да може да значајно ослаби перформансе система

- Кеш се додаје као међуслој између регистара (тј. процесора) и радне меморије
  - брзина између регистара и радне меморије
  - величина између регистара и радне меморије
- Ако је разлика превелика, додаје се више слојева кеш меморије

Хијерархија меморија



Примери брзина и величина кеш меморија

CPU Type	Pentium	Pentium Pro	Pentium II	Pentium III	Pentium 4
CPU speed	233MHz	200MHz	450MHz	1.4GHz	3.6GHz
L1 cache speed	4.3ns (233MHz)	5.0ns (200MHz)	2.2ns (450MHz)	0.71ns (1.4GHz)	0.28ns (3.6GHz)
L1 cache size	16KiB	32KiB	32KiB	32KiB	20KiB
L2 cache type	onboard	on-chip	on-chip	on-die	on-die
CPU/L2 speed ratio		1/1	1/2	1/1	1/1
L2 cache speed	15ns (66MHz)	5ns (200MHz)	4.4ns (225MHz)	0.71ns (1.4GHz)	0.28ns (3.6GHz)
L2 cache size	varies	256K	512K	512K	1M
CPU bus speed	66MHz	66MHz	100MHz	133MHz	800MHz
Memory bus speed	60ns (16MHz)	60ns (16MHz)	10ns (100MHz)	7.5ns (133MHz)	1.25ns (800MHz)

Принцип рада кеша

- Основни принцип рада је захватање података из радне меморије унапред (*prefetch*), пре него што заиста затребају процесору
  - ако је успешно предвиђено који ће подаци бити потребни процесору у блиској будућности, њих ће процесор читати из кеша а не из меморије
  - тиме се значајно подижу перформансе система

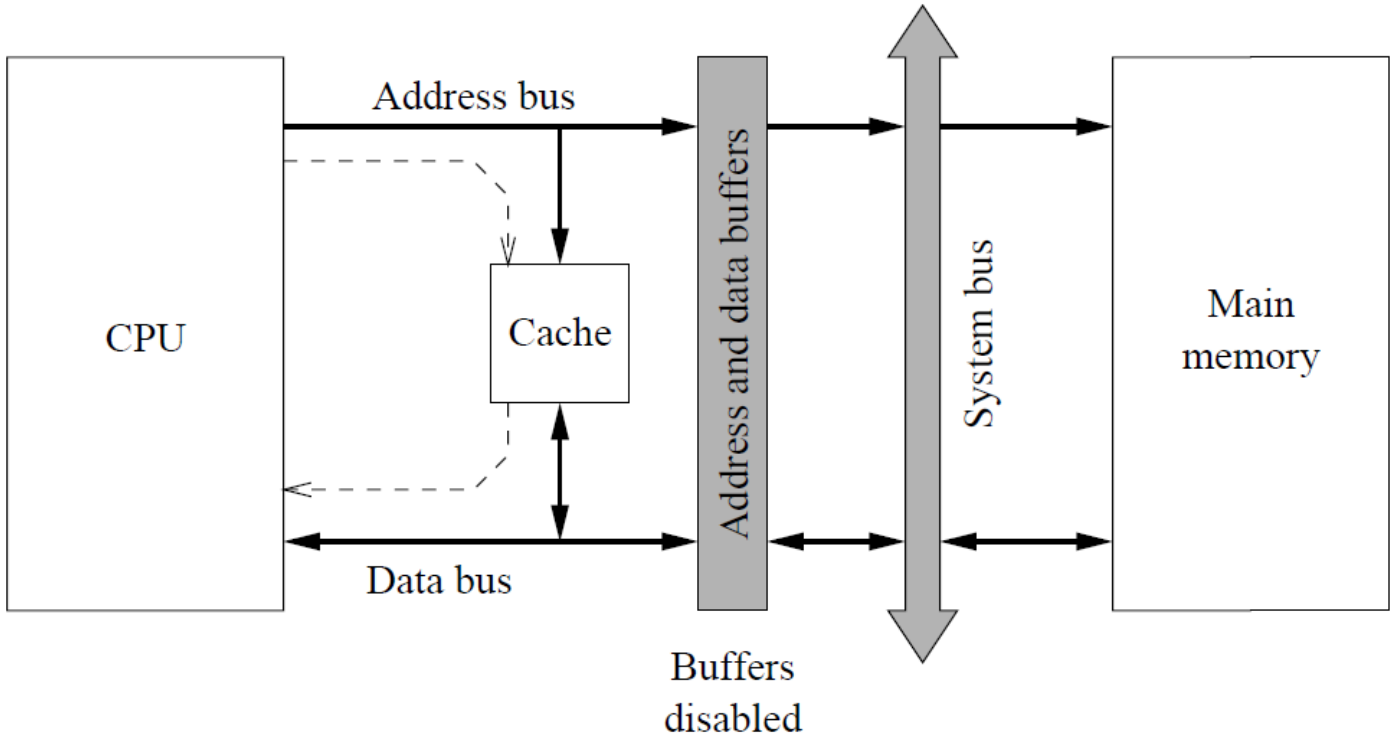
Основне операције кеша

- Кеш се користи у случају две основне меморијске операције
  - читање и
  - писање

- У оба случаја постоје по две варијанте
  - када су подаци присутни у кешу
    - тзв. *погодак* (*hit*)
  - када подаци нису присутни у кешу
    - тзв. *промашај* (*miss*)

Читање у случају поготка

- Ако се потребни подаци налазе у кешу, онда се одатле и читају



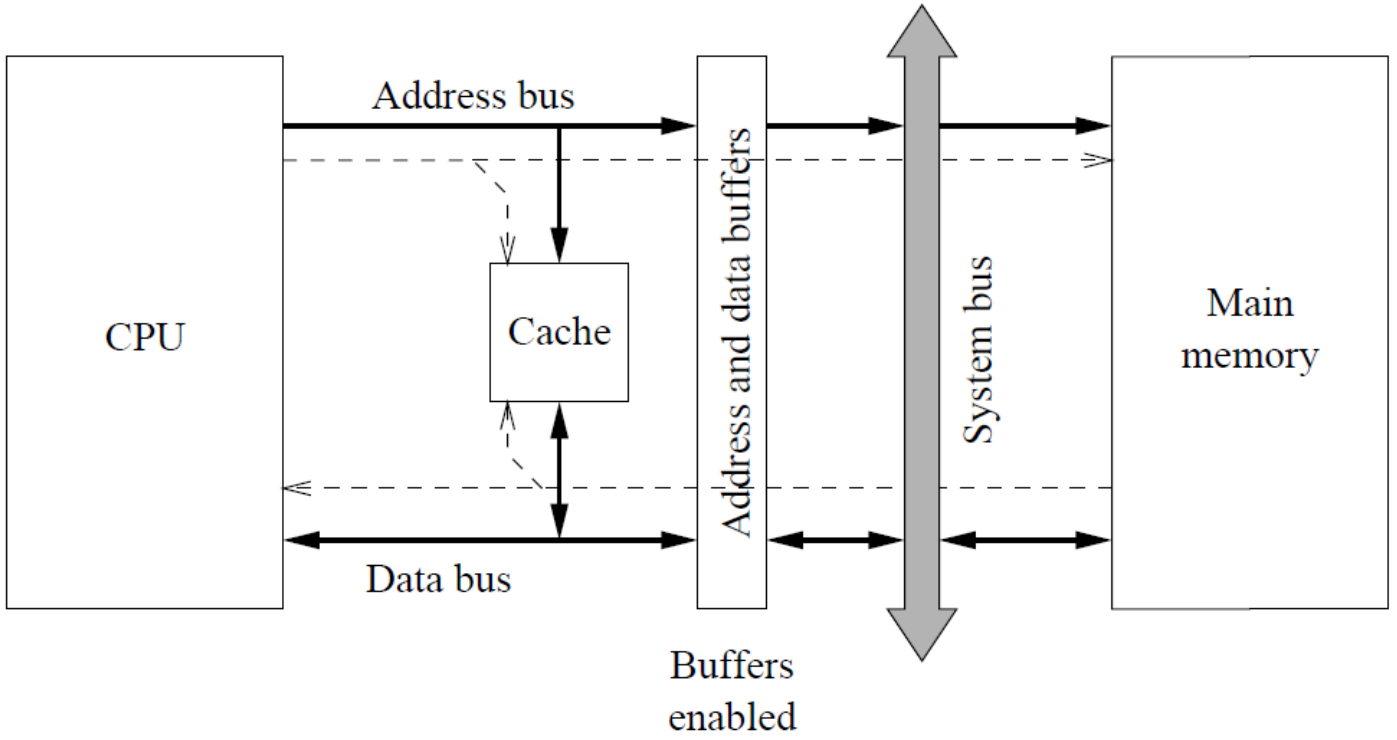
Читање у случају поготка (2)

- У случају поготка, линије адресе и података према меморији се блокирају
- Размена информација се одвија искључиво са кешом

- Читање са поготком је значајно брже од читања из меморије без примене кеша

Читање у случају промашаја

- Ако се потребни подаци не налазе у кешу, онда се читају из меморије и истовремено уписују у кеш



Читање у случају промашаја (2)

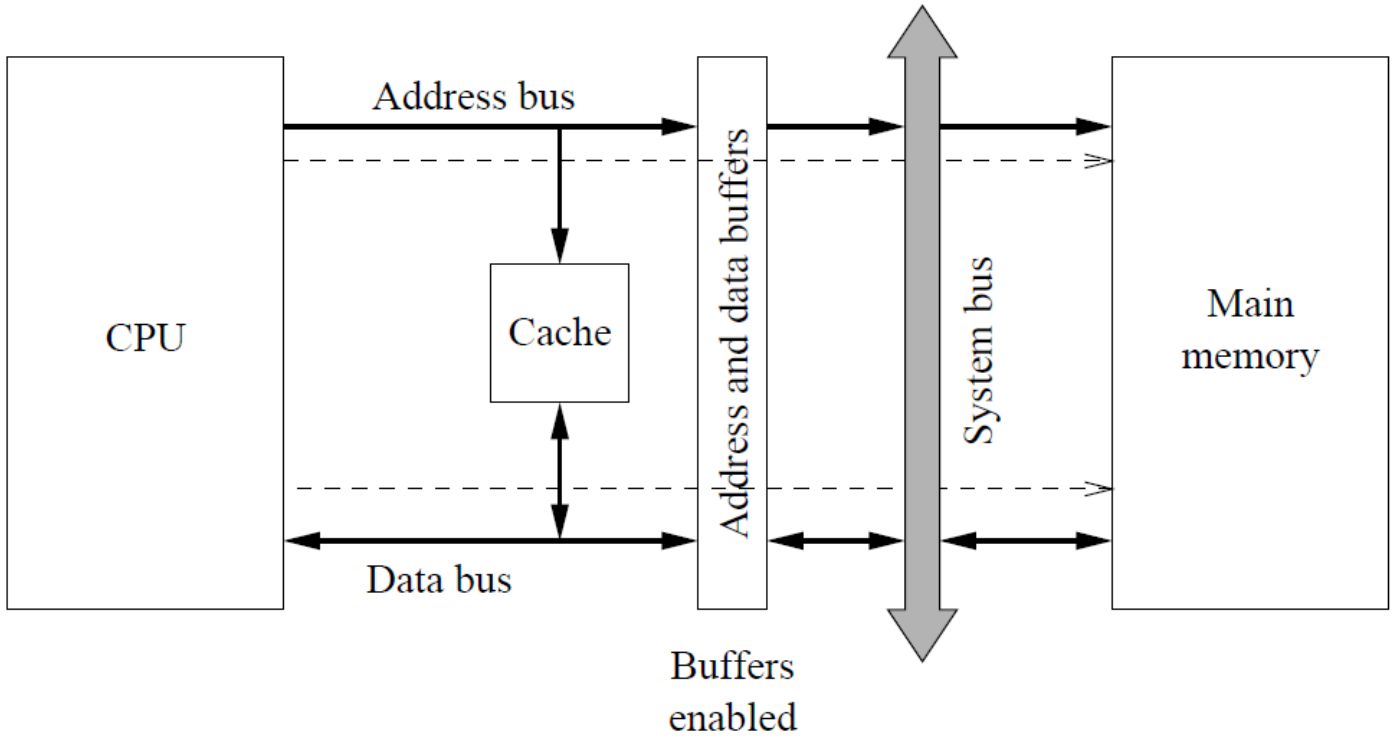
- У случају промашаја, линије адресе и података према меморији су активне
- Одвија се уобичајено (као да нема кеша) читање података из меморије
  - додатно се прочитани подаци уписују и у кеш
- Читање са промашајем је нешто спорије од читања из меморије без примене кеша
  - због неопходног проверавања да ли податак постоји у кешу или не

Перформансе кеша

- Поред брзине рада кеш меморије постоје и додатне мере перформанси
  - ***степен погодака*** (*hit rate*, *hit ratio*) је коефицијент који показује колико често се тражени подаци проналазе у кешу
  - ***степен промашаја*** (*miss rate, miss ratio*) показује колико често се тражени подаци не проналазе у кешу
    - (степен погодака + степен промашаја) = 1
  - ***време поготка*** (*hit time*) је време потребно да се податак прочита ако је у кешу
    - обухвата и време потребно да се провери да ли је податак у кешу
  - ***цена промашаја*** (*miss penalty*) је време потребно да се установи да податак није у кешу и да се читање преусмери на меморију

Писање у случају промашаја

- У случају промашаја подаци се уписују само у меморију, зато што не постоје у кешу

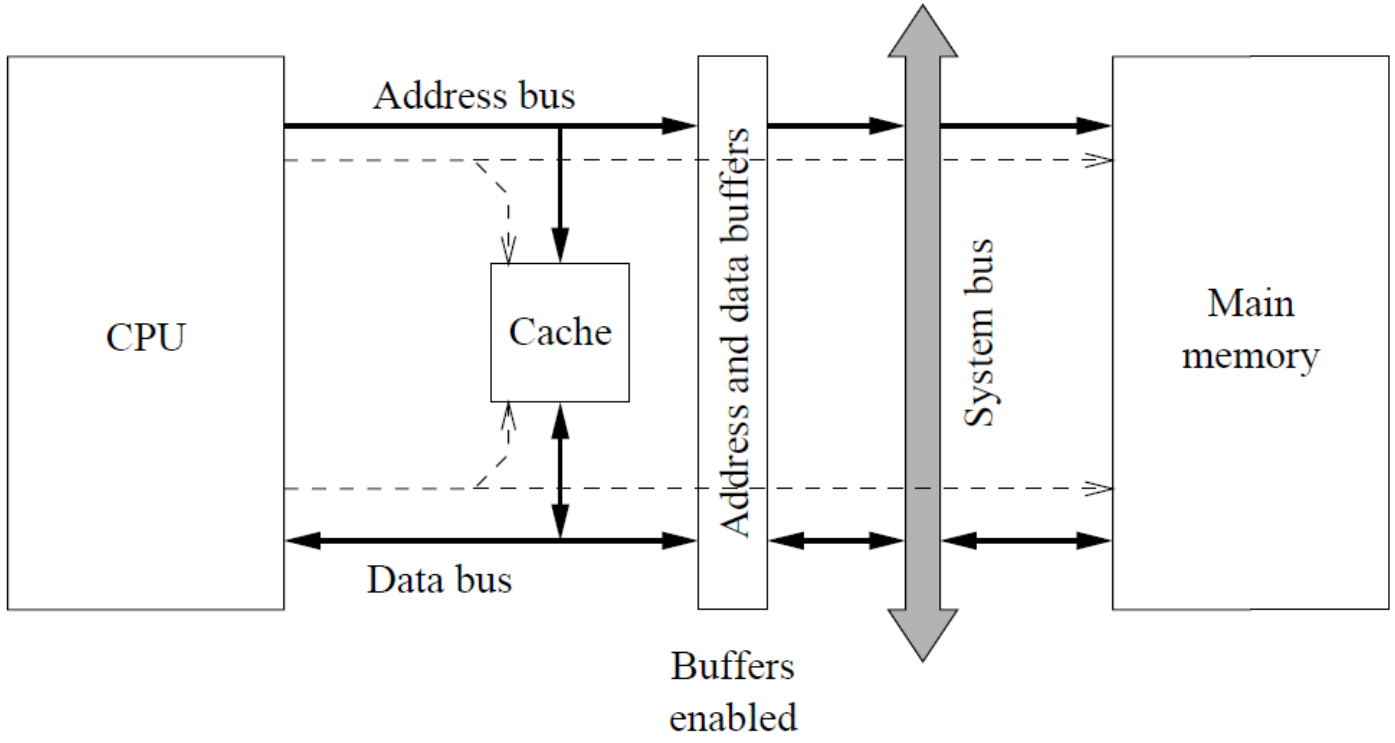


Писање у случају поготка

- У случају поготка постоје две основне могућности:
  - писање се обавља само у кеш или
  - писање се обавља и у кеш и у меморију

Писање у случају поготка (2)

- Случај писања и у меморију и у кеш



Значајна питања

- Како се установљава да ли је података у кешу или није?
- Ако податак није у кешу, како се у њега уписује?
- Колико података се уписује у једној операцији?
- Шта се дешава ако нема више места?
- Како се претпоставља чему ће процесор приступати у блиској будућности?

Зашто кеш ради?

- Практичан пример: увећавање свих елемената матрице (нпр. *double*) за *K*:

```
for(int i=0; i<M; i++)
   for(int j=0; j<N; j++)
      X[i][j] = X[i][j] + K;
```

Зашто кеш ради? (2)

- Први фактор за успешну примену кеша је *поновљена употреба* истих података или делова кода
  - Наредба у петљи се понавља *M*x*N* пута
  - Ако је наредба записана у кешу, њено извршавање је значајно убрзано зато што се чита из брзог кеша а не из споре меморије

Зашто кеш ради? (3)

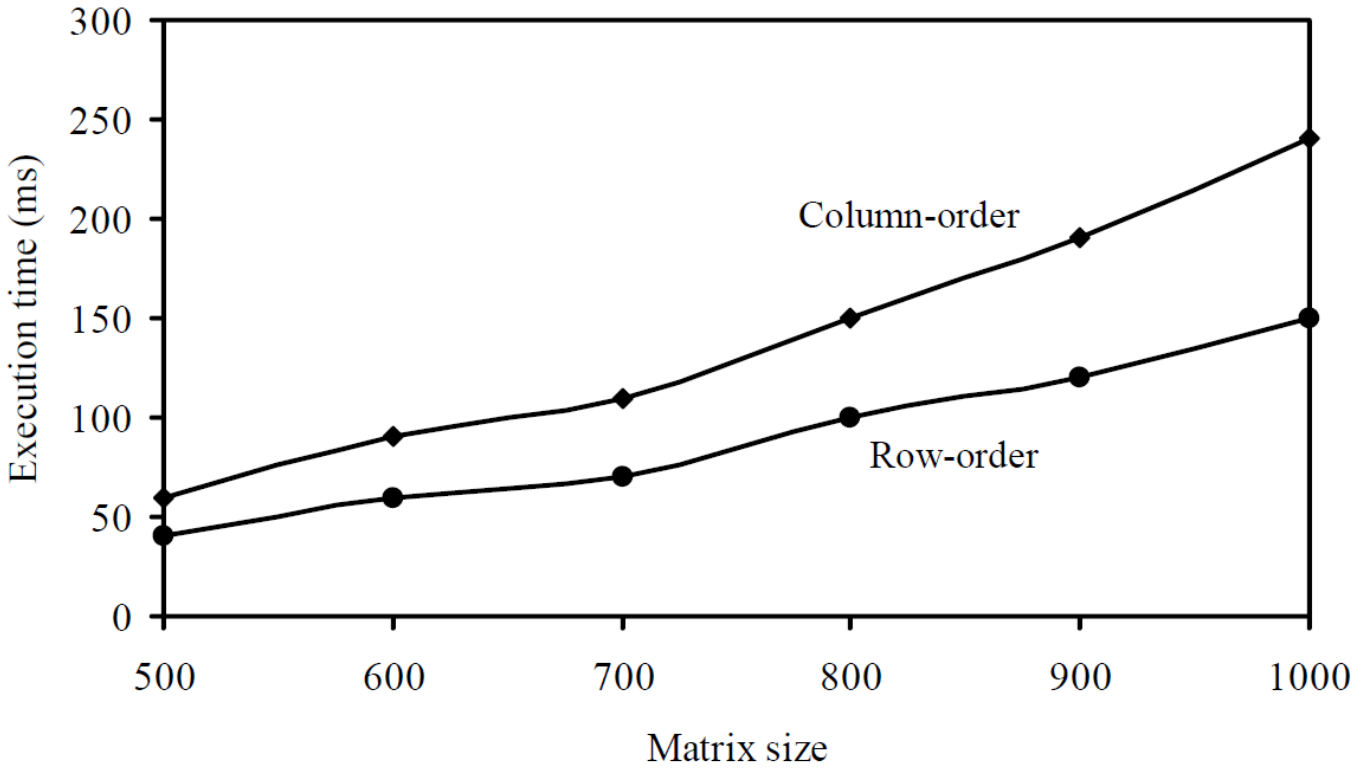
- Други фактор је планско пуњење кеша подацима и инструкцијама пре него што их процесор затражи
  - Планско пуњење подиже перформансе из два разлога:
    - маскира кашњење спорије главне меморије
    - пуњење се одвија у блоковима, а пренос блокова података је вишеструко бржи него пренос појединачних података

Зашто кеш ради? (4)

- Посматрамо пример и разматрамо само читање, без писања
  - При читању податка *X*[0][0] долази до промашаја
    - Приступа се главној меморији
    - Податак се чита и истовремено уписује у кеш
    - Осим њега, у кеш се уписују и *X*[0][1], *X*[0][2] и *X*[0][3]
      - (претпостављамо да се ради са блоковима по 32 бајта)
  - Наредне три итерације се читају из кеша, без употребе главне меморије
  - Иако нема поновљене употребе, добитак у перформансама се остварује захваљујући успешном предвиђању који ће подаци бити потребни

Зашто кеш ради? (5)

- Дијаграм илуструје трајање извршавања претходног примера кода
  - када се ради ред по ред (*row-order*)
    - читање података унапред доприноси
  - када се замене две петље и ради по колонама (*column-order*)
    - читање података унапред скоро да не доприноси ефикасности



Понашање програма

- У случају већине програма се испољава бар један од фактора за успешност кеша
  - понављање и/или
  - предвидиво приступање подацима

Принцип локалности

- Принцип локалност реферисања тврди да програми имају тенденцију да у неком датом периоду времена реферишу само неки подскуп података и инструкција и то често уз понављање

- Разликујемо
  - просторну локалност и
  - временску локалност

Просторна локалност

- Програми имају тенденцију да податке и инструкције користе секвенцијално
  - Локални подаци у функцијама су записани на стеку, блиско једни другима
  - Сложени подаци заузимају секвенцијалне области у меморији
  - Већину времена инструкције се читају и извршавају секвенцијално
    - скокови ремете секвенцијалност, али између два скока обично је више инструкција

Временска локалност

- Програми имају тенденцију да поновљено користе податке и инструкције у одређеним периодима времена
  - Типичан пример су петље
    - исте инструкције се извршавају више пута
    - исте локалне променљиве се користе више пута
    - неки делови сложених података се користе више пута

Иницијално стање кеша

- Непосредно по укључивању рачунара кеш је празан
  - тј. садржи отпатке
- Не постоји посебна процедура иницијалног пуњења кеша
- Примењују се исти поступци пуњења и коришћења као и у уобичајеном раду кеша

Пресликавање адреса

- Када је потребно приступити садржају неке меморијске локације, најпре се проверава да ли је већ у кешу или није
- Ако јесте потребно је установити где се у кешу налази
  - тј. израчунати његову адресу у кешу

- Овај поступак се назива *пресликавање адреса*
  - Адреса меморије се пресликава у адресу кеша

Функције пресликавања

- Пресликавање адреса се одвија неком од функција пресликавања:
  - непосредно пресликавање
  - асоцијативно пресликавање
  - скуп-асоцијативно пресликавање

(о начинима пресликавања ће бити речи нешто касније)

Величина блока

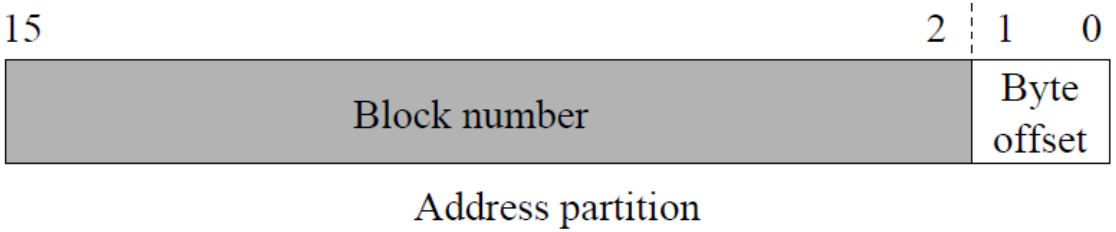
- Ако податак није у кешу, чита се из меморије
- Том приликом се уписује у кеш
- Због локалности података, из меморије се чита и у кеш уписује не само један податак него већи блок

- Величина блока може бити променљива или константна
  - нпр. Код процесора *Intel Pentium* и *PowerPC* блокови су фиксне величине: 32 бајта

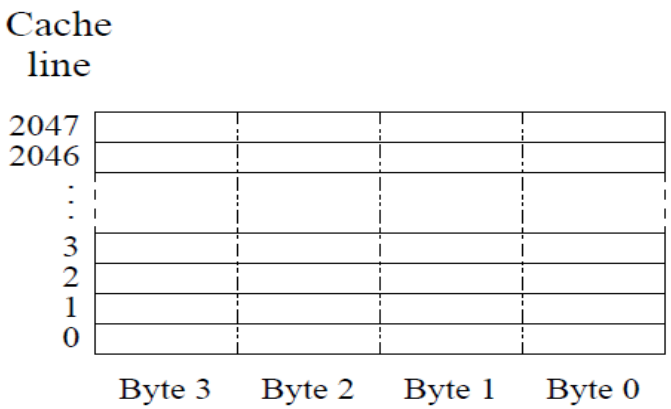
Поравнавање блока

- Ради једноставности али и ефикасности читања, често се прибегава поравнању блокова
  - Ако је величина блока B бајтова, обично се адресе равнају по почецима блокова
  - Нижих $b = \log_2 B$ битова адресе чине *положај у блоку*
  - Преостали виши битови чине *број блока*
- И садржај кеша се дели по блоковима величине B бајтова
- Блокови у кешу се називају *линије кеша*

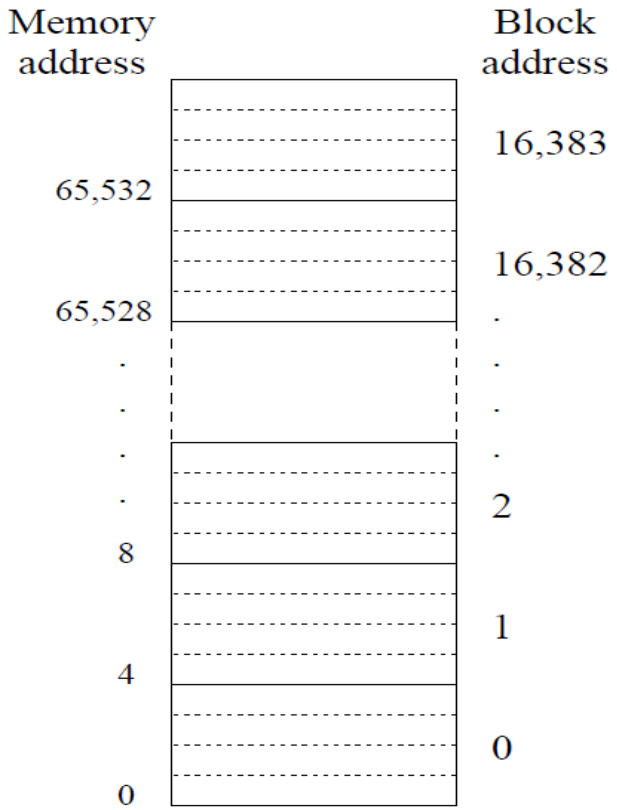
Пример поравнвања блокова величине 4 бајта



Address partition



Cache memory

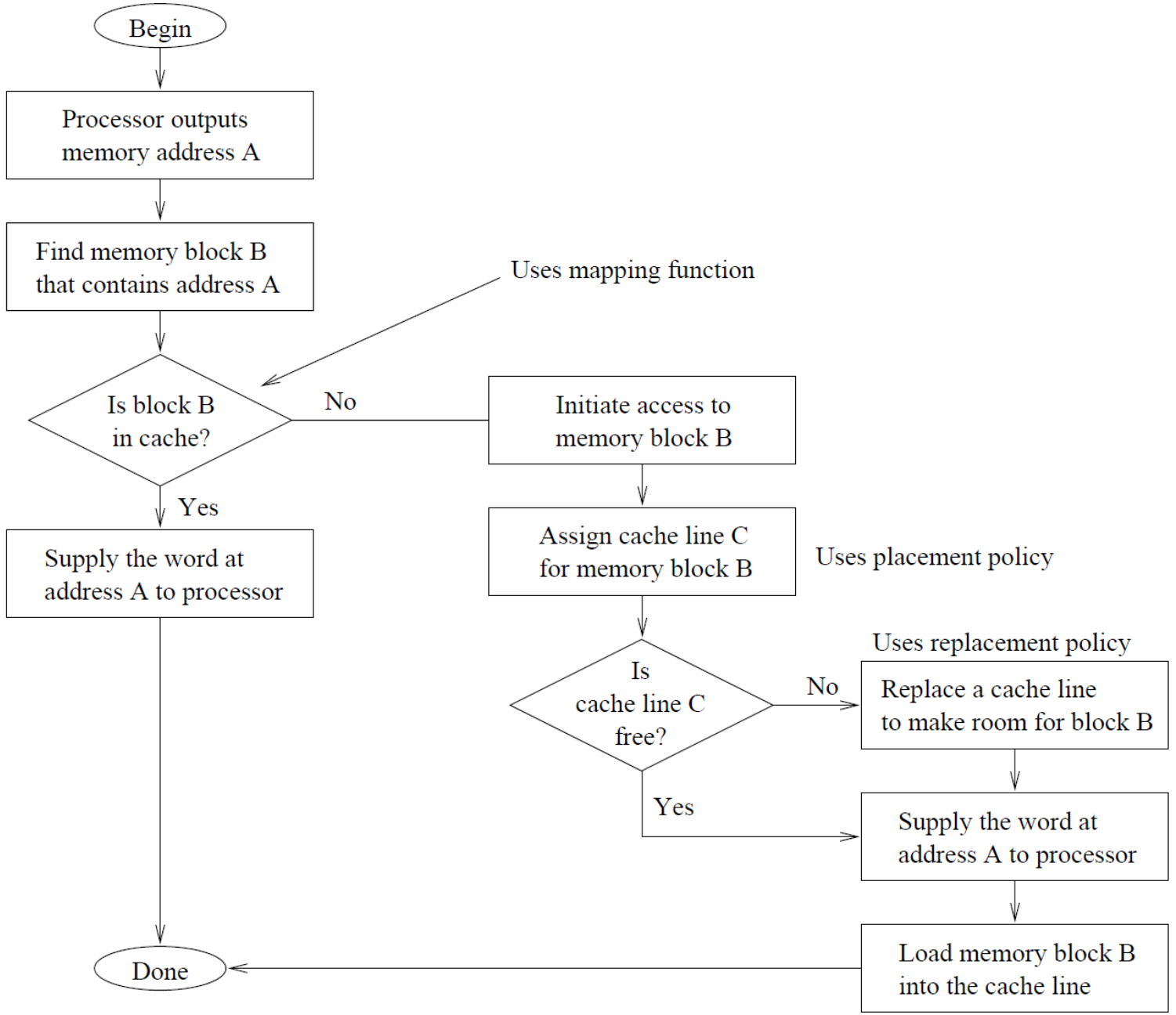


Main memory

Пуњење кеша

- Линија кеша у коју ће се уписати прочитан блок одређује се функцијом пресликавања
- Ако та линија кеша није слободна, потребно је одабрати линију кеша која је кандидат за замену
  - Ако функција пресликавања израчунава фиксну линију онда не постоји избор
  - Ако функција пресликавања израчунава скуп линија кеша, онда се примењује нека политика замењивања за избор линије кеша

Алгоритам читања



Врсте функција пресликавања

- Непосредно пресликавање
  - одређује тачно једну линију кеша за сваки меморијски блок
  - једноставна имплементација
  - ниска флексибилност може да ослаби перформансе

- Асоцијативно пресликавање
  - нема ограничења - свака линија кеша може бити коришћена за било који меморијски блок
  - максимална флексибилност
  - веома сложена имплементација

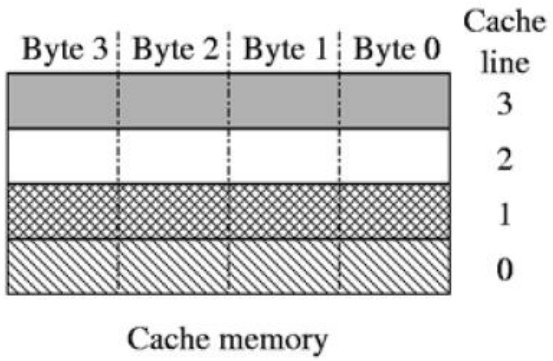
- Скуп-асоцијативно пресликавање
  - одређује се скуп линија кеша за сваки меморијски блок
  - компромис између претходне две врсте пресликавања
  - најчешће се примењује

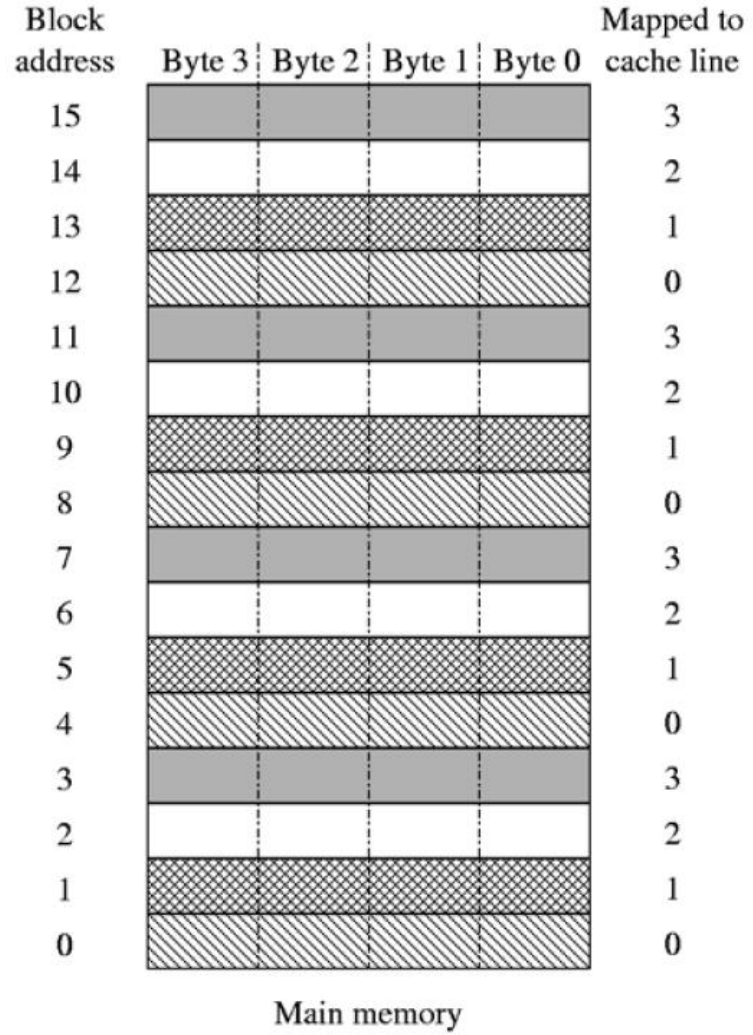
Непосредно пресликавање

- Пресликавањем се одређује тачно једна линију кеша за сваки меморијски блок
  - Обично се користи функција $c = i \bmod C$
    - c – линија кеша
    - i – меморијски блок
    - C – број линија кеша (величина кеша у линијама)

Пример непосредног пресликавања

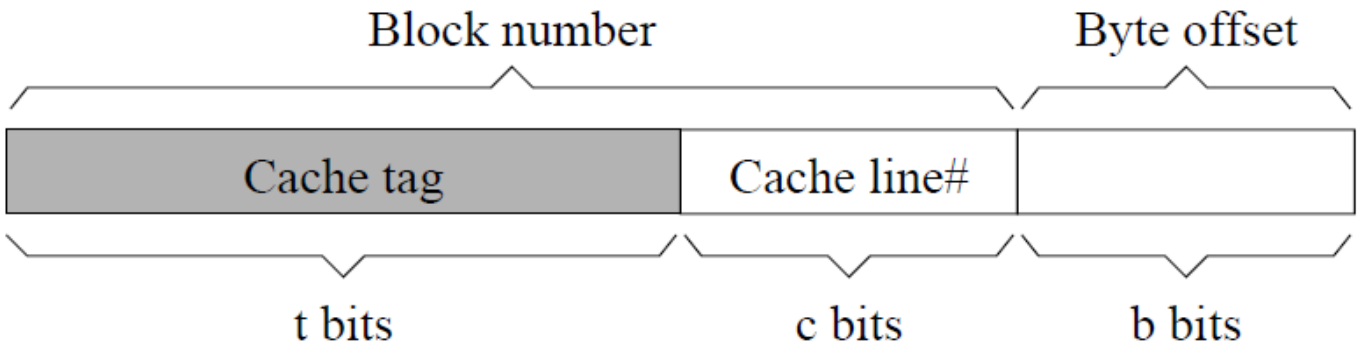
- Меморија 64*B*
- Кеш 16*B*
- Блок 4*B*





Делови адресе

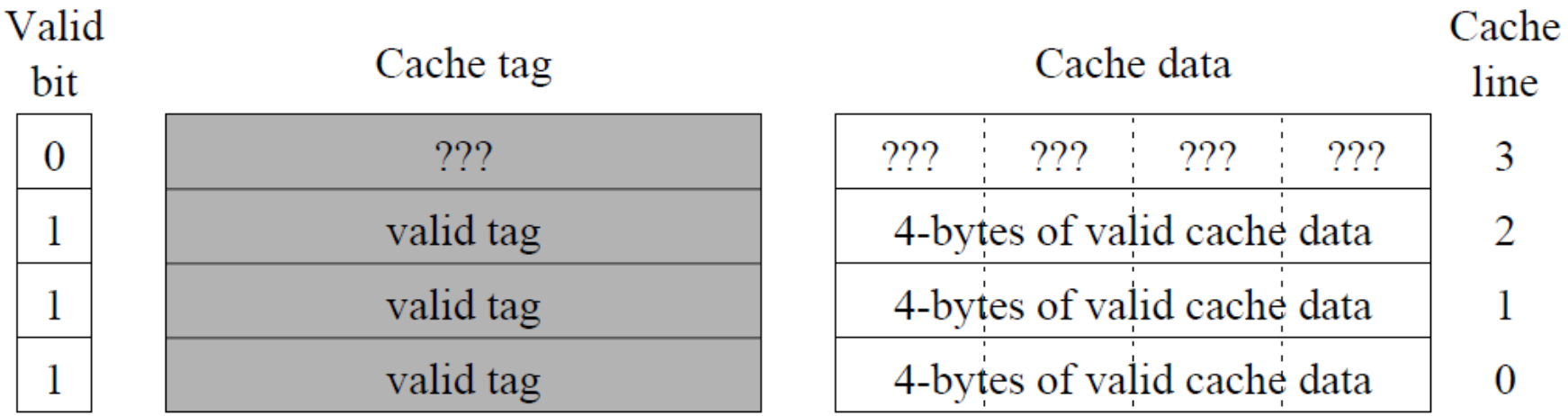
- Делови меморијске адресе су:
  - најнижих $b = \log_2 B$ битова адресе чине *положај у блоку*
  - наредних $c = \log_2 C$ битова чине *линију кеша*
  - највиших t битова чине *ознаку кеша*



Садржај кеша

- За сваку линију кеша морају постојати следећи подаци:
  - садржај кеша
    - *B* бајтова који представљају садржај линије кеша
    - копија пресликаног меморијског блока
  - бит исправности
    - означава да ли је садржај линије исправан
    - 1 означава исправан а 0 неисправан садржај
    - иницијално је 0
  - ознака кеша (енгл. *tag*)
    - *t* битова који означавају ком меморијском блоку одговара тренутни садржај линије кеша

Садржај кеша (2)



Пример непосредног пресликавања

- Процесор *Intel Pentium*:
  - адресни простор је 4*GiB*
  - кеш са непосредним пресликавањем 256*KiB*
  - величина линије кеша је 32*B*

  - положај у блоку је дужине b = 5 битова
  - број линија кеша је 256*KiB* / 32*B* = 8192 = 8*Ki*
  - адреса линије кеша је дужине $c = \log_2 8192$ = 13 битова
  - ознака кеша је преосталих 32-5-13 = 14 битова

Поступак пресликавања

- Да би се пронашла одговарајућа линија кеша, примењује се функција пресликавања по модулу
- Ако је линија кеша исправна и ознака кеша одговара меморијском блоку, то значи да садржај линије кеша већ одговара датом меморијском блоку
- Ако блок није у кешу, онда се чита из меморије и уписује у одговарајућу линију кеша
  - ако је линија кеша била попуњена, њен садржај се претходно обрађује у складу са политиком писања

Пример непосредног пресликавања

- Нека је величина кеша 4 линије x 4 бајта
- Нека се приступа редом меморијским адресама:
  - 0, 16, 0, 32, 0, 32, 0, 16, 0, 16, 0, 16
- Колики је степен погодака?

- Адресе 0, 16 и 32 одговарају блоковима 0, 4 и 8
- Свим овим блоковима одговара линија кеша 0
- Степен погодака је 0

- Ово је пример најгорег могућег сценарија
- Случај да се садржај линије кеша замењује пре него што се ијединој поново реферише назива се *растресање кеша*

Пример непосредног пресликавања (2)

Block accessed	Hit or miss	Cache line 0	Cache line 1	Cache line 2	Cache line 3
0	Miss	Block 0	???	???	???
4	Miss	Block 4	???	???	???
0	Miss	Block 0	???	???	???
8	Miss	Block 8	???	???	???
0	Miss	Block 0	???	???	???
8	Miss	Block 8	???	???	???
0	Miss	Block 0	???	???	???
4	Miss	Block 4	???	???	???
0	Miss	Block 0	???	???	???
4	Miss	Block 4	???	???	???
0	Miss	Block 0	???	???	???
4	Miss	Block 4	???	???	???

Пример непосредног пресликавања (3)

- Нека је величина кеша 4 линије x 4 бајта
- Нека се приступа редом меморијским блоковима:
  - 0, 7, 9, 10, 0, 7, 9, 10, 0, 7, 9, 10
- Колики је степен погодака?

- Ово је пример најбољег могућег сценарија
- Свака линија кеша се пуни само по једанпут

Пример непосредног пресликавања (4)

Block accessed	Hit or miss	Cache line 0	Cache line 1	Cache line 2	Cache line 3
0	Miss	Block 0	???	???	???
7	Miss	Block 0	???	???	Block 7
9	Miss	Block 0	Block 9	???	Block 7
10	Miss	Block 0	Block 9	Block 10	Block 7
0	Hit	Block 0	Block 9	Block 10	Block 7
7	Hit	Block 0	Block 9	Block 10	Block 7
9	Hit	Block 0	Block 9	Block 10	Block 7
10	Hit	Block 0	Block 9	Block 10	Block 7
0	Hit	Block 0	Block 9	Block 10	Block 7
7	Hit	Block 0	Block 9	Block 10	Block 7
9	Hit	Block 0	Block 9	Block 10	Block 7
10	Hit	Block 0	Block 9	Block 10	Block 7

Асоцијативно пресликавање

- Свака линија кеша може бити коришћена за било који меморијски блок
- Максимална флексибилност
- Веома сложена имплементација
  - Потенцијални проблеми са перформансама

Пример асоцијативног пресликавања

- Нека је величина кеша 4 линије x 4 бајта
- Нека се приступа редом меморијским адресама:
  - 0, 16, 0, 32, 0, 32, 0, 16, 0, 16, 0, 16
- Колики је степен погодака?

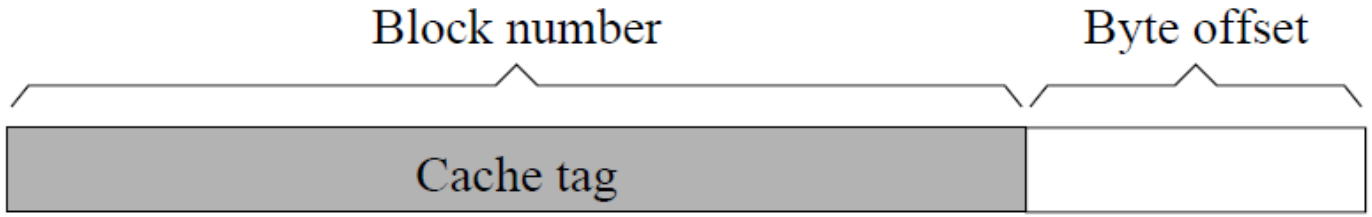
- Сваки блок може да користи било коју линију кеша
  - Блок 0 се записује у првој слободној линији (нпр. 0)
  - Блок 4 у наредној слободној линији (нпр. 1)
  - Блок 8 у наредној слободној линији (нпр. 2)
- Степен погодака је 9/12 = 0.75
  - То је највиши степен који се може постићи на датом примеру

Пример асоцијативног пресликавања (2)

Block accessed	Hit or miss	Cache line 0	Cache line 1	Cache line 2	Cache line 3
0	Miss	Block 0	???	???	???
4	Miss	Block 0	Block 4	???	???
0	Hit	Block 0	Block 4	???	???
8	Miss	Block 0	Block 4	Block 8	???
0	Hit	Block 0	Block 4	Block 8	???
8	Hit	Block 0	Block 4	Block 8	???
0	Hit	Block 0	Block 4	Block 8	???
4	Hit	Block 0	Block 4	Block 8	???
0	Hit	Block 0	Block 4	Block 8	???
4	Hit	Block 0	Block 4	Block 8	???
0	Hit	Block 0	Block 4	Block 8	???
4	Hit	Block 0	Block 4	Block 8	???

Делови адресе

- Делови меморијске адресе су:
  - најнижих $b = \log_2 B$ битова адресе чине *положај у блоку*
  - свих преосталих t битова чине *ознаку кеша*



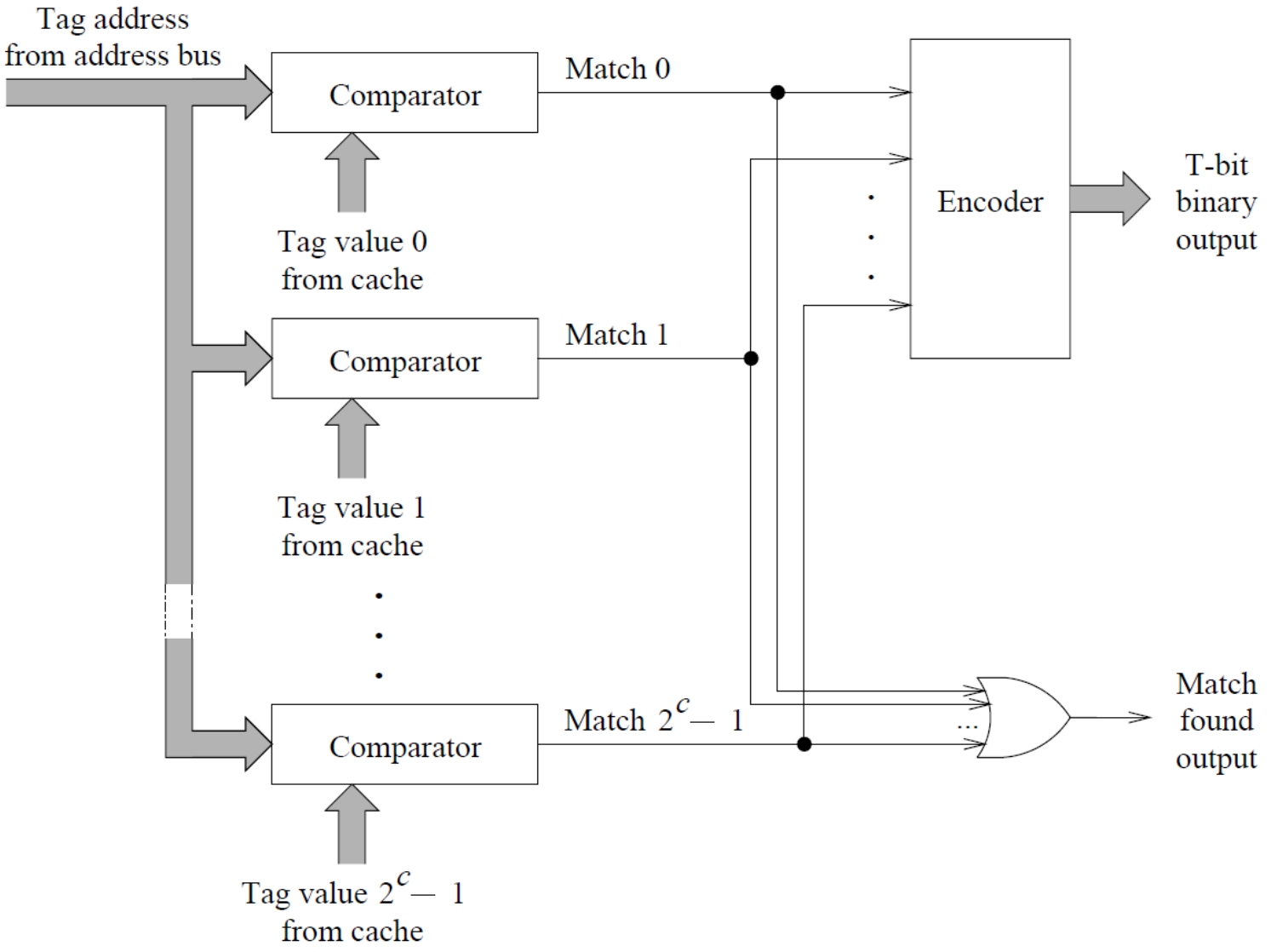
Проблеми са асоцијативним пресликавањем

- Дужина ознаке кеша је већа него код непосредног пресликавања
  - ово није најважнији проблем али
  - има утицаја на количину потребних података за линију кеша

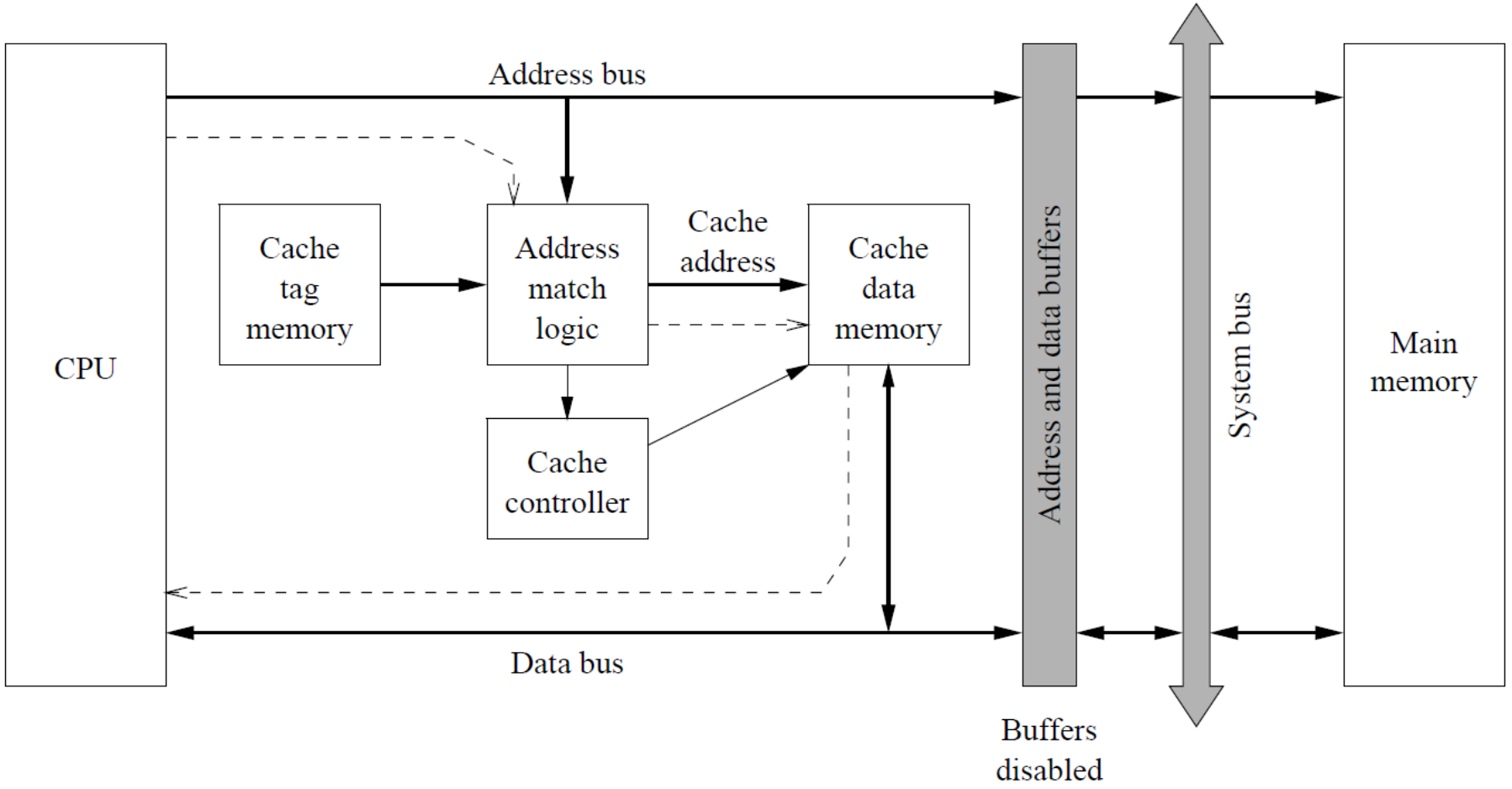
- Провера да ли је меморијски блок већ записан у некој линији кеша је значајно сложенија
  - уместо да се проверава само једна линија кеша, потребно је да се ознаке *свих* исправних линија кеша упореде са ознаком датог блока
  - то захтева 2^c поређења дужине $t+c$ битова

- Последица је да је имплементација сложенија и скупља него код непосредног пресликавања

Поређење ознака код асоцијативног пресликавања



Читање података из кеша са асоцијативним пресликавањем

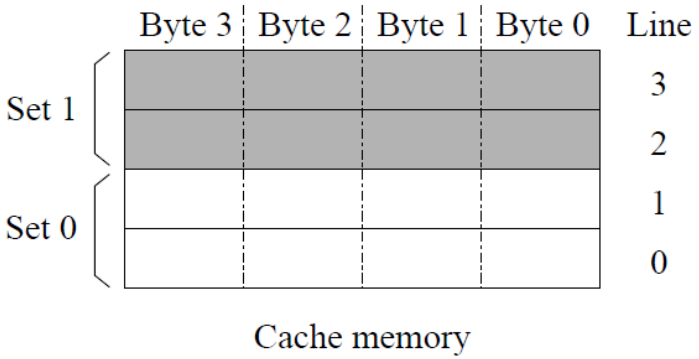


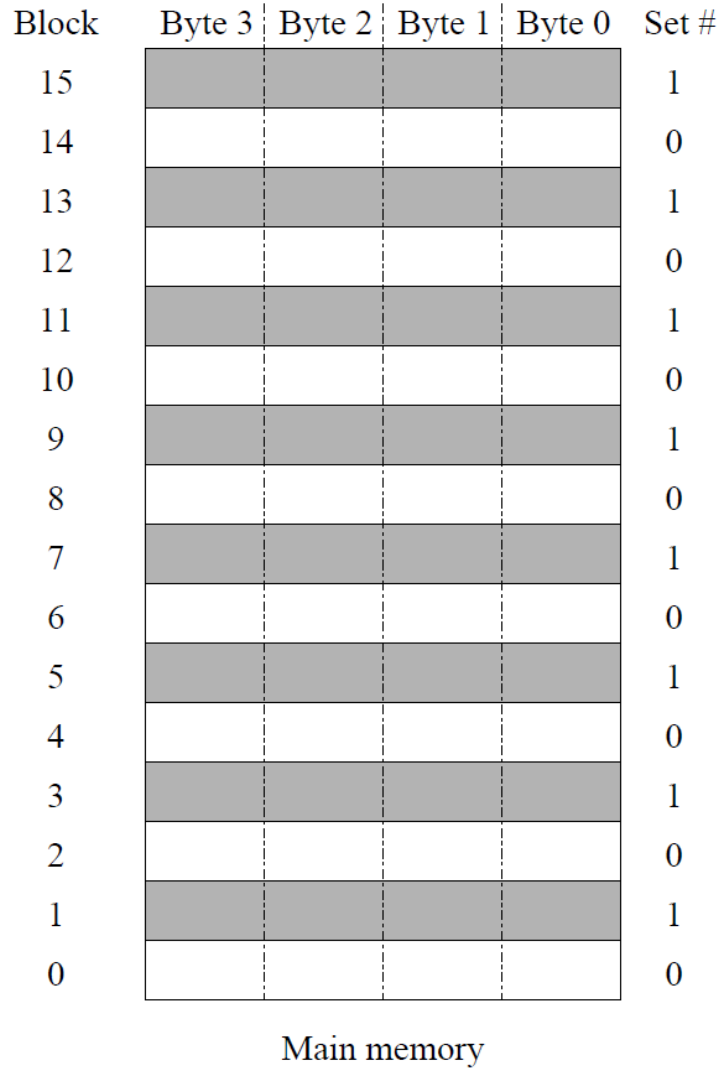
Скуп-асоцијативно пресликавање

- Компромис између непосредног и асоцијативног пресликавања
- Пресликавање се одвија слично непосредном пресликавању, али се за један меморијски блок одређује скуп линија кеша уместо само једне линије кеша
  - Величина скупа је обично неки мањи степен броја 2
    - 2, 4, 8 или 16
  - енгл. “*4-way associative mapping*” (за скупове вел. 4)

Пример скуп-асоцијативног пресликавања

- Меморија 64*B*
- Блок 4*B*
- Кеш 16*B*
  - 2 скупа x 2 линије x 4*B*





Пример скуп-асоцијативног пресликавања

- Нека је величина кеша 4 линије x 4 бајта
- Нека се приступа редом меморијским адресама:
  - 0, 16, 0, 32, 0, 32, 0, 16, 0, 16, 0, 16
- Колики је степен погодака?

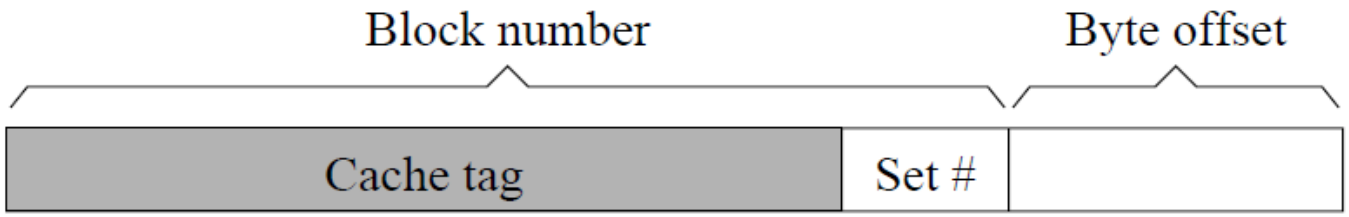
- Сваки од блокова 0,4 и 8 се пресликава у скуп линија 0
  - Блок 0 се записује у првој слободној линији скупа 0 (нпр. 0)
  - Блок 4 у наредној слободној линији скупа 0 (нпр. 1)
  - При првом приступању блоку 8 све линије скупа 0 су већ заузете:
    - примењује се *политика замењивања* како би се одабрала линија чији ће се задржај заменити
    - на пример, ако се замењује линија која дуже није коришћена, то ће бити линија блока 4
    - када се поново буде приступало блоку 4, поступа се слично
- Степен погодака је 8/12 = 0.67

Пример скуп-асоцијативног пресликавања (2)

Block accessed	Hit or miss	Set 0		Set 1	
		Cache line 0	Cache line 1	Cache line 0	Cache line 1
0	Miss	Block 0	???	???	???
4	Miss	Block 0	Block 4	???	???
0	Hit	Block 0	Block 4	???	???
8	Miss	Block 0	Block 8	???	???
0	Hit	Block 0	Block 8	???	???
8	Hit	Block 0	Block 8	???	???
0	Hit	Block 0	Block 8	???	???
4	Miss	Block 0	Block 4	???	???
0	Hit	Block 0	Block 4	???	???
4	Hit	Block 0	Block 4	???	???
0	Hit	Block 0	Block 4	???	???
4	Hit	Block 0	Block 4	???	???

Делови адресе

- Делови меморијске адресе су:
  - најнижих $b = \log_2 B$ битова адресе чине *положај у блоку*
  - наредних $s = \log_2 S$ битова одређују *скуп линија кеша*
  - највиших t битова чине *ознаку кеша*
    - употребљава се као код асоцијативног пресликавања



Предности скуп-асоцијативног пресликавања

- Предности у односу на непосредно су сличне као и предности асоцијативног
  - омогућавање вишег степена погодака
    - и у случају најгорег сценарија
    - и у просечном случају
- Предности у односу на асоцијативно су
  - мања дужина ознаке кеша
  - мањи број кандидата које је потребно проверавати
    - уместо 2^c поређења по t битова
    - потребно је 2^s поређења по t-s битова
- Слабости
  - нешто нижи степен погодака него у случају асоцијативног пресликавања

Политике замењивања

- Политика замењивања се примењује ради одабира линије кеша чији ће садржај бити замењен садржајем новом меморисјког блока
- Зависи од примењеног пресликавања
  - у случају непосредног пресликавања не постоје политике замењивања зато што нема избора
  - у случају скуп-асоцијативног пресликавања
    - политика замењивања најдуже некоришћене линије кеша
    - политика замењивања псеудо-најдуже-некоришћене линије
    - политика произвољног замењивања
    - FIFO – прва прочитана се прва замењује
    - LFU – најмање пута коришћена

Политика замењивања најдуже некоришћене линије кеша

- (енгл. *least recently used – LRU*)

- Идеално би било да се замени онај меморијски блок који најдуже неће бити поново коришћен
- “Предвиђање” се заснива на принципу локалности
- Замењује се садржај оне линије кеша која најдуже није била коришћена

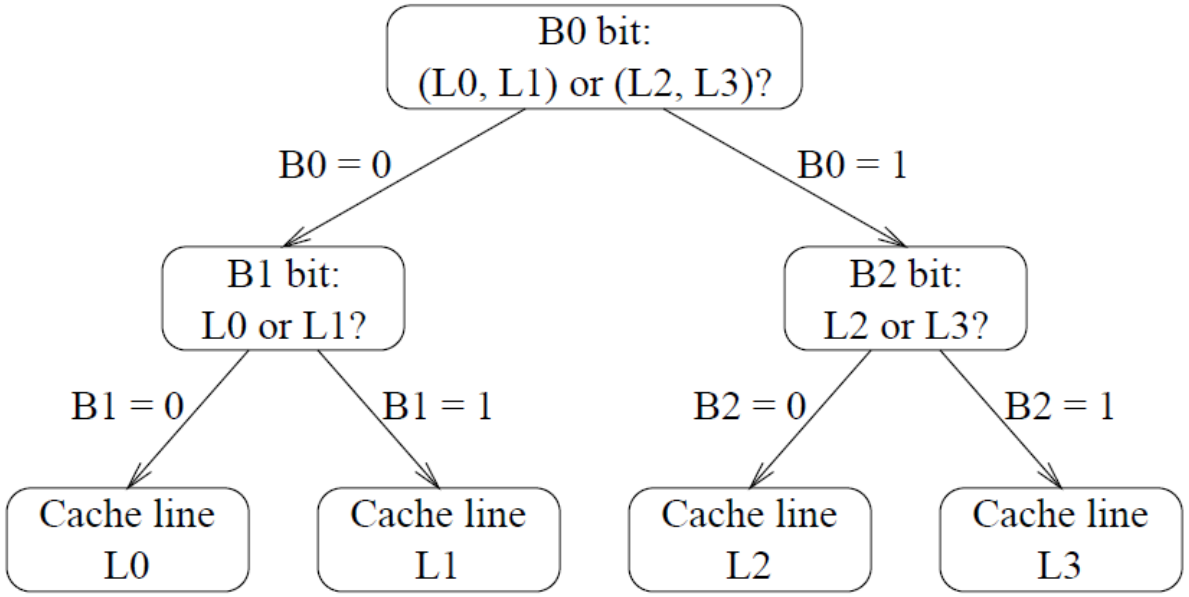
Имплементација

- За скупове величине 2 довољан је један бит по скупу, за означавање последње коришћене линије
- За веће скупове се праве одговарајући коначни аутомати
  - За скуп величине 4
    - 4! = 24 могућих кобинација редоследа
    - потребно је 5 битова по скупу линија
  - За скуп величине 8
    - 8! = 40320 могућих комбинација редоследа
    - потребно је 16 битова по скупу линија

Политика замењивања псеудо-најдуже-некоришћене линије кеша

- У пракси се поједностављује имплементација политике “најдуже некоришћене” линије кеша:
  - све линије кеша у скупу се деле на две групе
  - помоћу једног бита се означава којој групи припада линија кеша која је последња коришћена у скупу
  - свака група се даље дели на подгрупе
- Мањи број битова и једноставнији алгоритам
  - за скуп величине *W* линија потребно је укупно *W*-1 битова (у пуној примени потребно $\log_2 W$)
- Умањена је и прецизност

Политика замењивања псеудо-најдуже-некоришћене линије кеша (2)



Политика произвољног замењивања

- Линија кеша из скупа се бира методом случајног избора
- Што су скупови већи, ова метода је прихватљивија
  - за скупове величине 2 даје око 10% слабији степен погодака
  - за веће скупове разлика постаје скоро безначајна
  - истраживања показују да се у неким случајевима понаша чак и боље него политика замењивања псеудо најмање коришћене линије

FIFO – прва прочитана се прва замењује

- Замењује се она линија кеша која је прва прочитана
- Релативно једноставна имплементација
  - скуп линија кеша се понаша као кружни бафер
  - за сваки скуп је потребан по један кружни бројач који означава која је линија последња попуњена (или која ће бити наредна попуњена / замењена)
- Релативно се ретко употребљава

LFU – најмање пута коришћена

- Замењује се она линија кеша која је најмање пута коришћена након попуњавања
- Нешто сложенија имплементација
  - за сваку линију кеша мора да се води по бројач
- Релативно се ретко употребљава

Политике писања

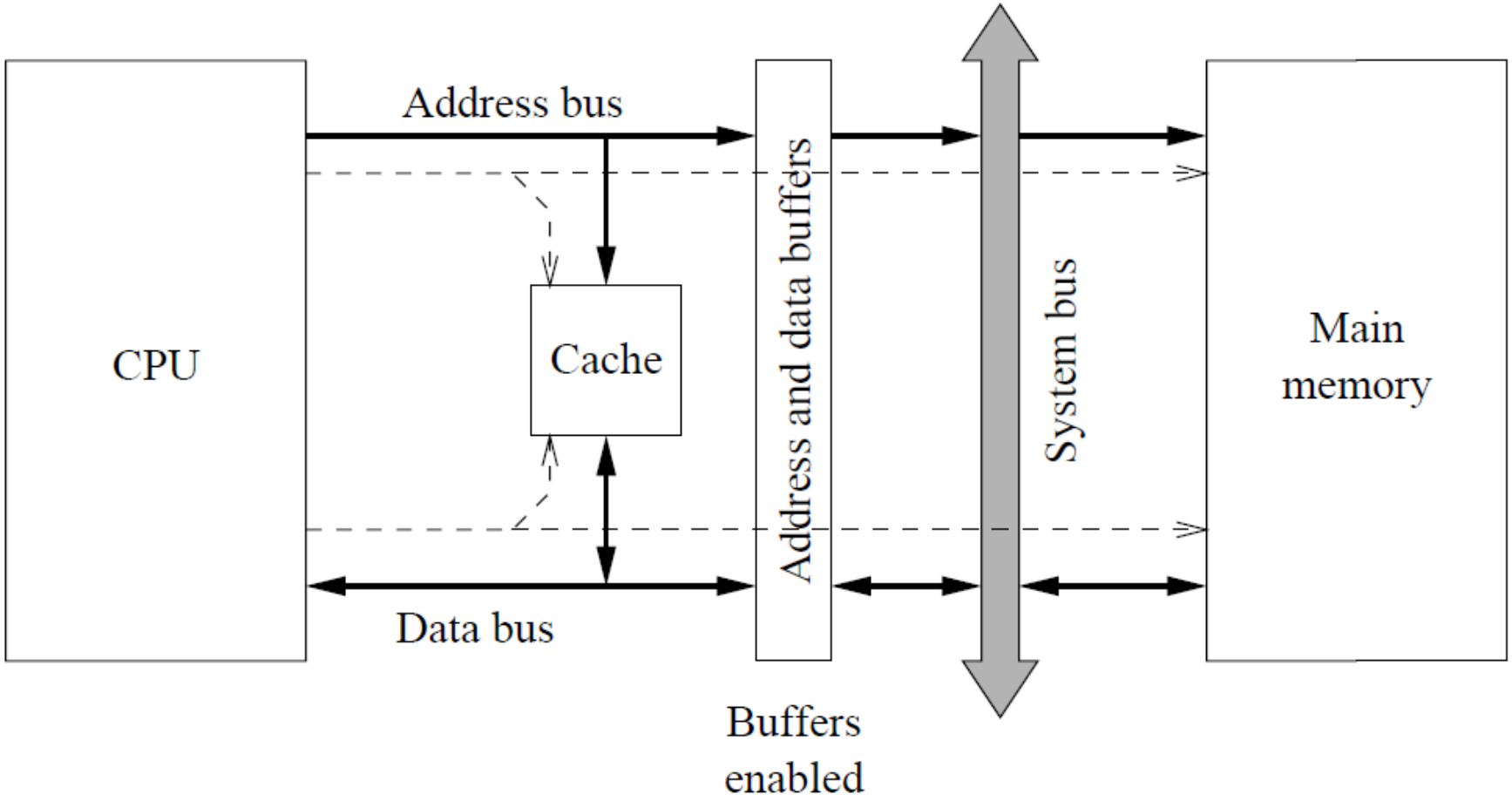
- Када се подаци мењају, мора се узети у обзир да постоје две копије података
  - копија у главној меморији и
  - копија у кешу

- Постоје две основне политике писања
  - писање се обавља само у кеш
    - тзв. политика *писање са преписивањем* (*write-back policy*)
  - писање се сваки пут обавља и у кеш и у меморију
    - тзв. политика *писање са пропуштањем* (*write-through policy*)

Политика писања са пропуштањем

- При писању се мењају обе копије података
  - мења се копија у кешу
  - мења се и копија у главној меморији

Операција писања у случају поготка и политике писања са пропуштањем



Политика писања са преписивањем

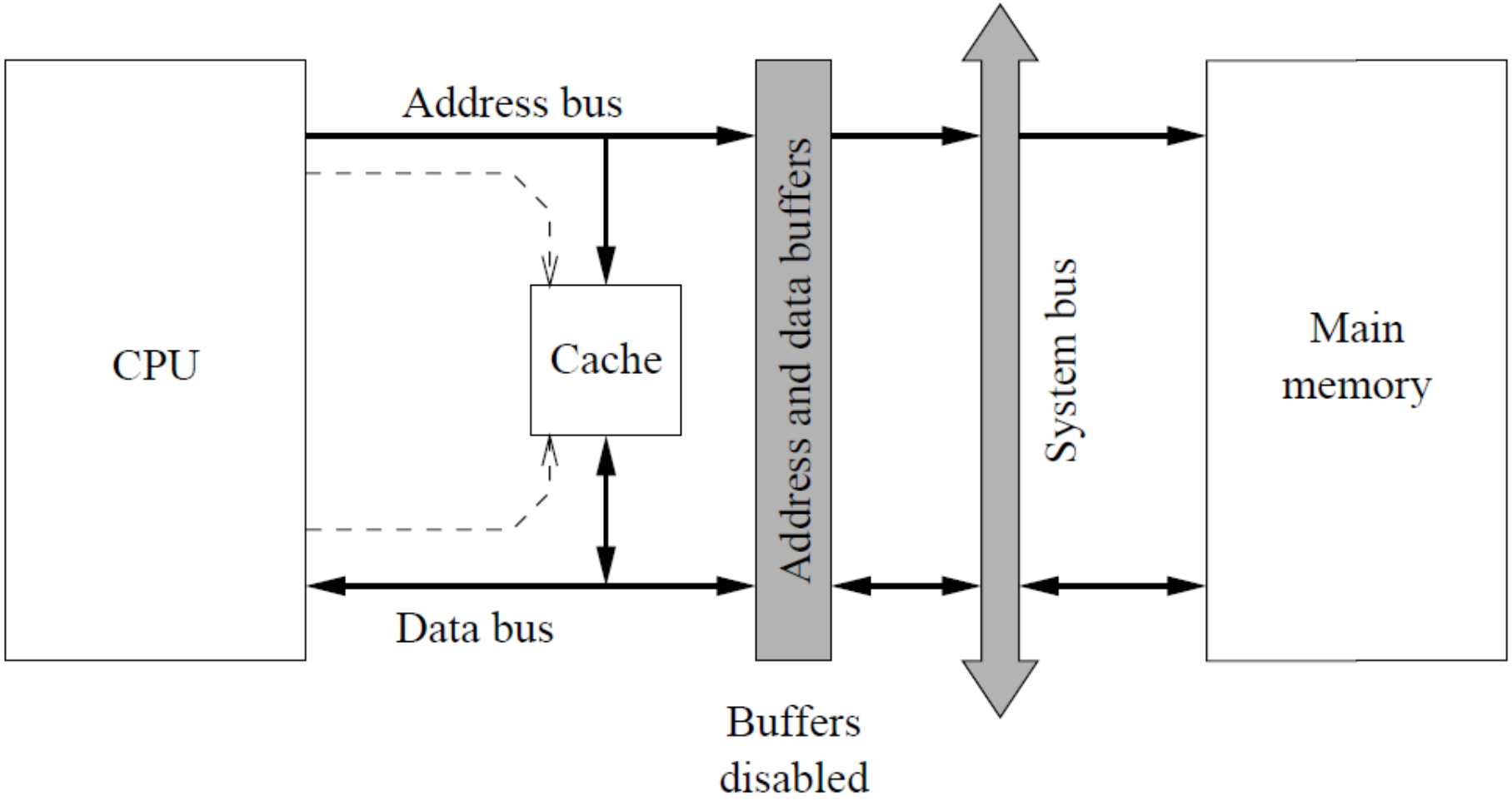
- При писању се мења само копија у кешу
- Копија у главној меморији се ажурира у тренутку замењивања линије кеша

- Замењивање је двоструко дуже него у случају политике писања са пропуштањем

Политика писања са преписивањем (2)

- Оптимизација се постиже евидентирањем да ли је било писања тако да се писање у главној меморији обавља само ако су подаци у кешу измењени
  - за сваку линију кеша се води додатни бит који означава да ли је садржај кеша различит од садржаја главне меморије
    - тзв. *dirty bit*
  - оваква оптимизација је прилично ефикасна зато што писања чине свега око 15% свих приступа меморији

Операција писања у случају поготка и политике писања са преписивањем



Однос политика писања

- Политика писања са пропуштањем:
  - добро је што су подаци у главној меморији и кешу увек усклађени
    - посебно важно у случају вишепроцесорских система
  - мана је што се при свакој операцији писања ангажује меморијска магистрала (и меморија)
    - пример: сумирање елемената низа

- Политика писања са преписивањем
  - добро је што се писање одвија само једанпут по линији кеша
  - мане:
    - замењивање је двоструко дуже
    - потребан је додатни бит по линији кеша

Бафери за писање

- Бафери за писање омогућавају додатну оптимизацију
- Посебно су корисни у случају политике писања са пропуштањем
  - зато што се ту писање одвија чешће него у случају писања са преписивањем
- Бафери за писање убрзавају рад тако што прикупљају податке за више операција писања, како би их касније одједанпут проследили меморији
- Пример:
  - *Intel Pentium* има 32-бајтни бафер за писање
  - стварно писање се дешава када се бафер попуни, али и у још неким случајевима

Додатни меморијски простор

- Све три функције пресликавања захтевају одређен додатни простор за ознаке
  - Додатни простор се смањује са нивоом асоцијативности
  - Додатни простор се смањује са повећавањем величине линија кеша

Пример додатног простора

- На пример, за неки 32-битни процесор
  - адресни простор 4*GiB*
  - кеш 32*KiB* са непосредним пресликавањем
  - величина блока 32B
    - 1024 линије
  - свака линија има
    - 32 x 8 = 256 битова података
    - 17 битова ознаке
    - 1 бит исправности
  - 18 / 256 = 7% додатних података
    - опционо још 1 бит усклађености (*dirty bit*)
      - зависно од политике писања

Пример додатног простора (2)

- На пример, за неки 32-битни процесор
  - адресни простор 4*GiB*
  - кеш 32*KiB* са непосредним пресликавањем
  - величина блока 4B
    - 8192 линије
  - свака линија има
    - 4 x 8 = 32 бита података
    - 17 битова ознаке
    - 1 бит исправности
  - 18 / 32 = 56% додатних података
    - опционо још 1 бит усклађености (*dirty bit*)
      - зависно од политике писања

Пример додатног простора (3)

- На пример, за неки 32-битни процесор
  - адресни простор 4*GiB*
  - кеш 32*KiB* са скуп-асоцијативним пресликавањем
    - величина скупа је 4 линије кеша
  - величина блока 32B
    - 1024 линије, 256 скупова
  - свака линија има
    - 32 x 8 = 256 битова података
    - 19 битова ознаке
    - 1 бит исправности
  - 20 / 256 = 7.8% додатних података
    - опционо још 1 бит усклађености (*dirty bit*)
      - зависно од политике писања

Пример додатног простора (4)

- На пример, за неки 32-битни процесор
  - адресни простор 4*GiB*
  - кеш 32*KiB* са асоцијативним пресликавањем
  - величина блока 32B
    - 1024 линије
  - свака линија има
    - 32 x 8 = 256 битова података
    - 27 битова ознаке
    - 1 бит исправности
  - 28 / 256 = 11% додатних података
    - опционо још 1 бит усклађености (*dirty bit*)
      - зависно од политике писања

Пример додатног простора (5)

Block size	Direct mapping (%)	4-way set-associative (%)	Fully associative (%)
32 bytes	7	7.8	11
4 bytes	56	62.5	97

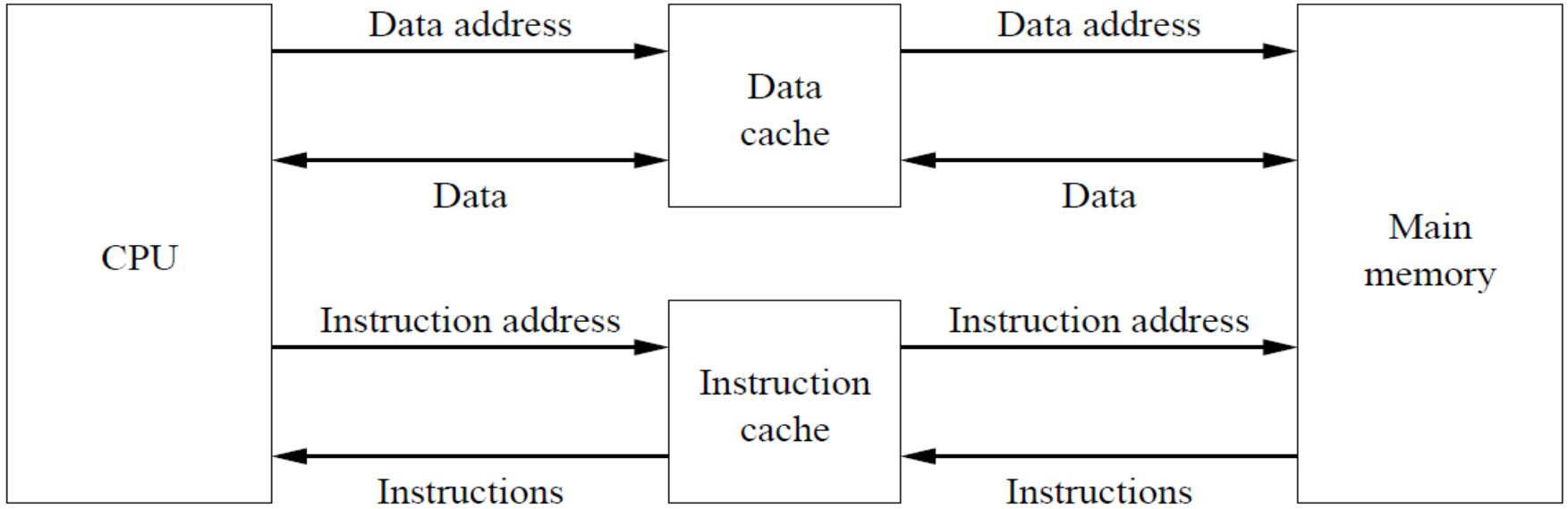
Врсте промашаја

- Неизбежни промашаји
  - први приступи неком меморијском блоку морају бити промашаји
- Промашаји услед капацитета
  - због мање величине кеша, мора доћи до замењивања садржаја линија кеша
  - могли би се избећи када би било више простора
- Промашаји услед судара
  - дешавају се у случају непосредног или скуп-асоцијативног пресликавања, када у кешу има простора, али не за конкретне блокове

Унапређења кеша

- Циљ
  - смањивање броја промашаја
  - умањивање утицаја промашаја на ефикасност
- Уводе се сложеније архитектуре кеша:
  - Раздвајање кеша података и кеша инструкција
  - Више нивоа кеша
  - Виртуални и физички кеш

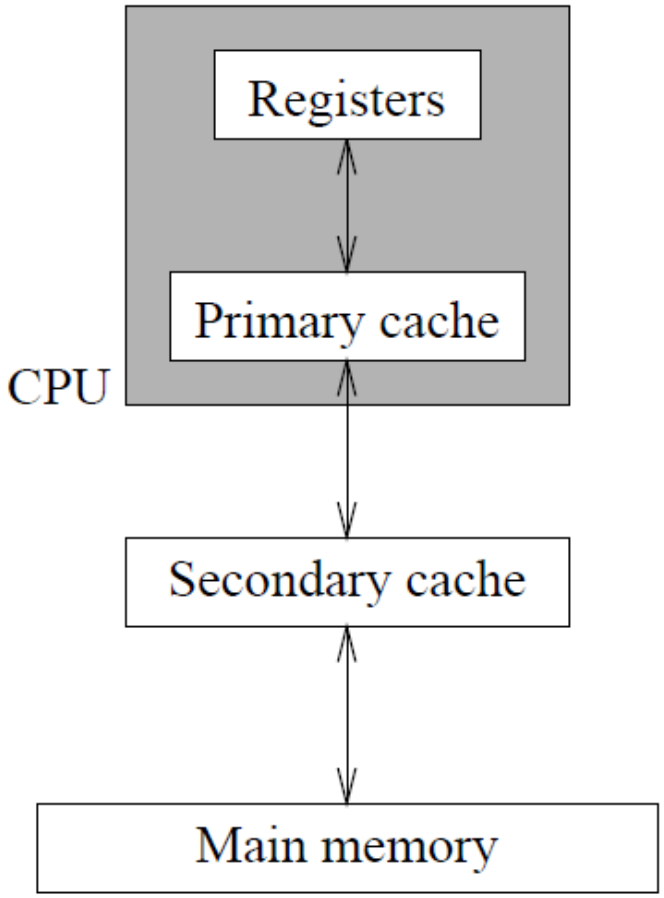
Раздвајање кеша података и кеша инструкција



Раздвајање кеша података и кеша инструкција (2)

- Основна идеја је да инструкције и подаци представљају различите врсте садржаја
  - на различит начин им се приступа
    - инструкције се обично не мењају
  - разликује се понављање употребе инструкција и података
- Ако се инструкције и подаци посматрају одвојено
  - локалност је тачнија и
  - будући приступи лакше предвидиви
- Мана је што се пропорција ова два кеша не може мењати динамички (према потреби)

Више нивоа кеша



Више нивоа кеша (2)

- Увођењем више нивоа кеша (*вишестепени кеш*) омогућава се
  - физичко лоцирање кеша на различитим местима
    - L1 кеш (кеш нивоа 1) на процесору, ближе регистрима
    - L2 кеш (кеш нивоа 2) ближе меморији
  - обезбеђивањем L2 кеша који је већи и са већим линијама смањује се цена промашаја кеша L1

Архитектуре вишестепеног кеша

- Експклузиван кеш
  - сваки блок је на највише једном нивоу кеша
  - смањено понављање значи већи простор
  - сваки ниво мора да има исту величину линија

- Инклузиван кеш
  - кеш вишег нивоа садржи блокове који су у кешу нижег нивоа
  - при замењивању блокова једног нивоа се врши писање само на следећем нивоу
  - кеш вишег нивоа може да користи веће линије

- Углавном инклузиван кеш
  - кеш вишег нивоа може али не мора да садржи блокове који су у кешу нижег нивоа

Рад вишестепеног кеша (инклузиван случај)

- Процесор полушава да приступи подацима у кешу L1
  - ако су подаци пронађени, они се читају из кеша L1 (*погодак кеша L1*)
  - ако нису (*промашај кеша L1*), подаци се морају читати из кеша L2 или из главне меморије

- У случају промашаја кеша L1, кеш контролер покушава да приступи подацима у кешу L2
  - ако су подаци пронађени у кешу L2, они се читају одатле (*погодак кеша L2*)
    - реч се прослеђује процесору
    - блок се уписује у кеш L1
  - ако нису (*промашај кеша L2*), подаци се читају из главне меморије
    - реч се шаље процесору
    - блок (или блокови различитих величина) се уписују у оба кеша

Перформансе вишестепеног кеша

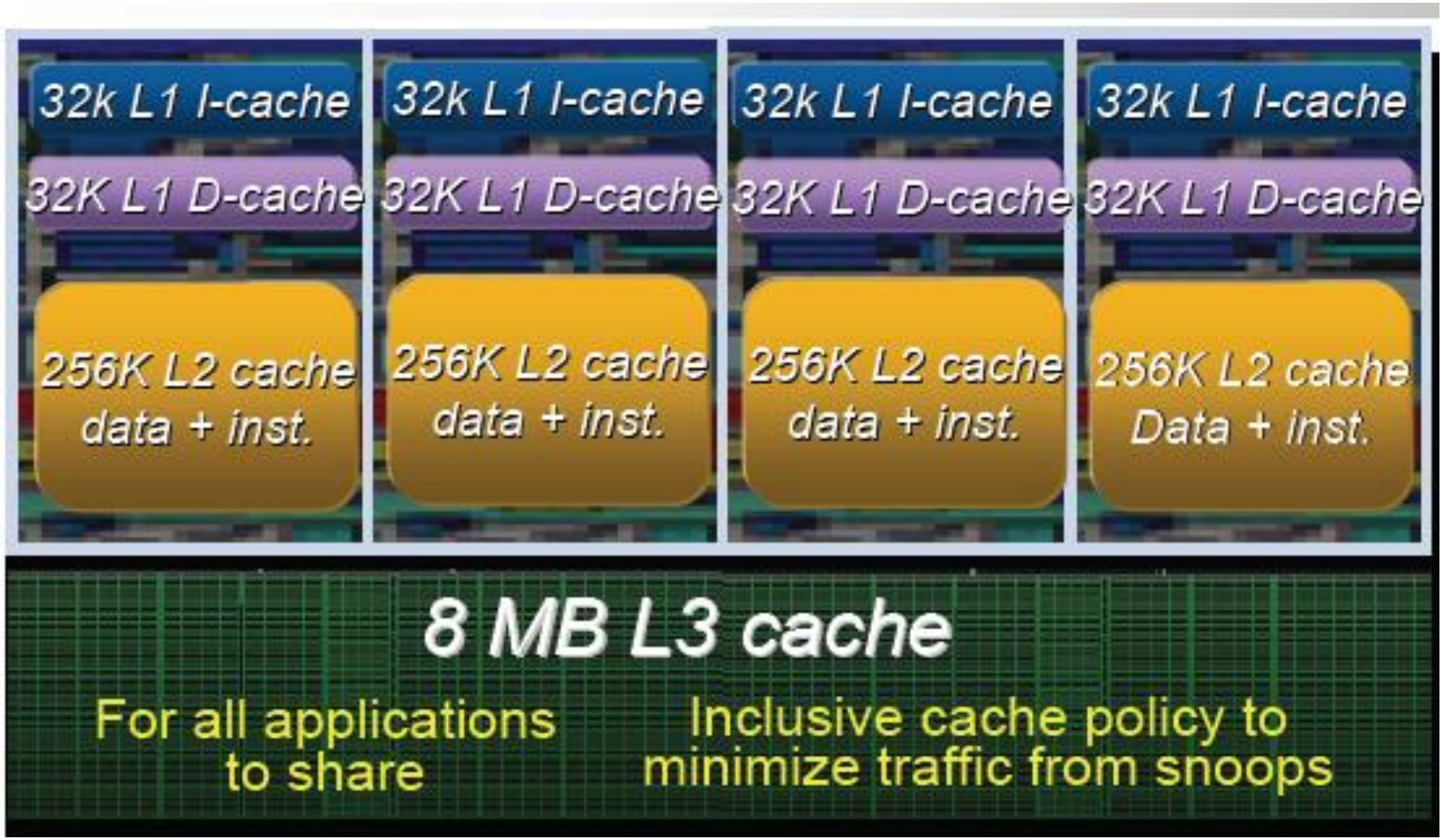
- Намена кеша L2 је да “ухвати” промашаје кеша L1
  - Ако је степен погодака кеша L1 90% и степен погодака кеша L2 такође 90%, онда је укупан степен промашаја (заступљеност приступа који морају да се обрате главној меморији) свега 1%

    (инклузиван случај)
    (идеализовано, за случај различитих величина линија)

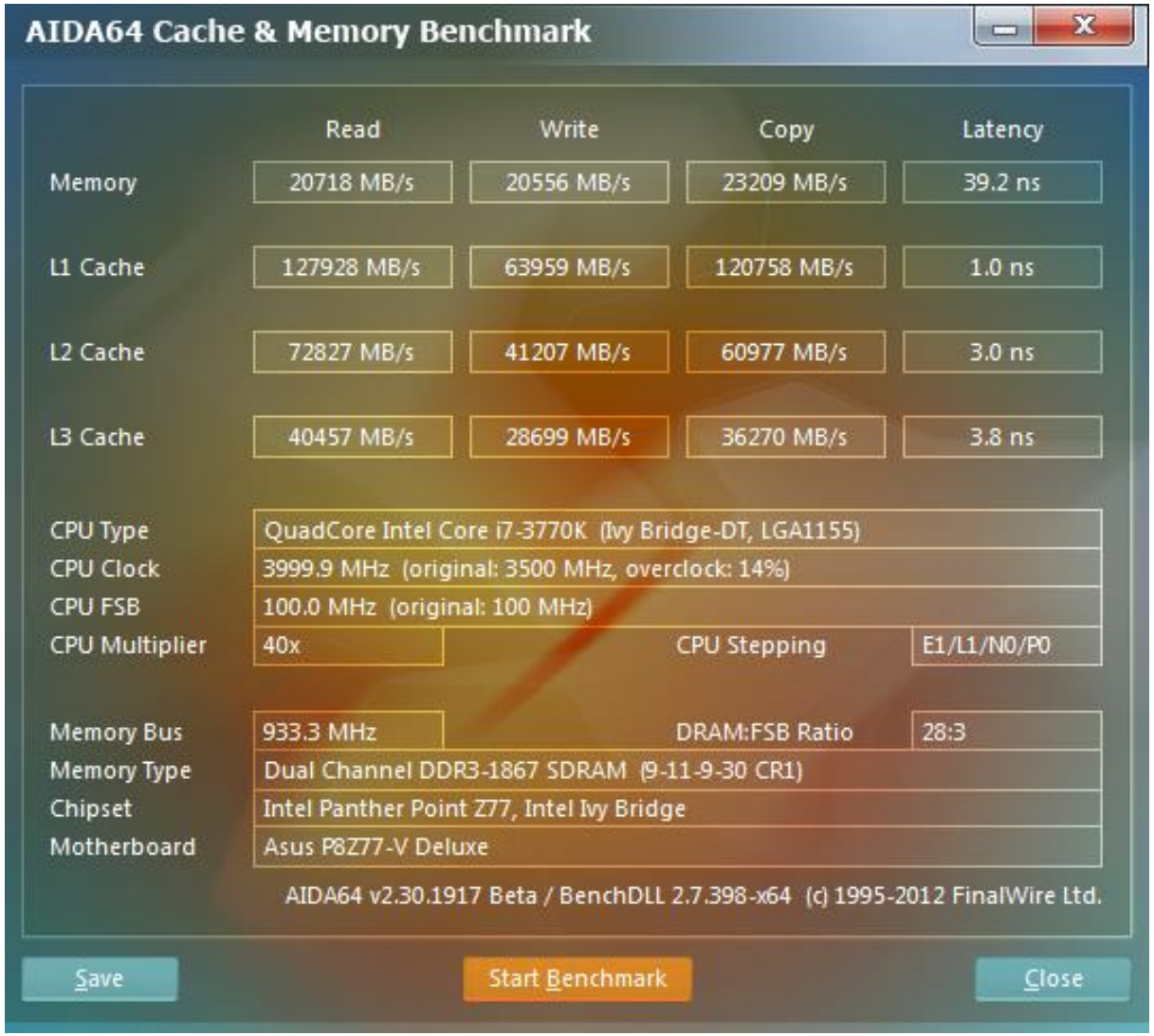
Пример савремене архитектуре

- Процесор *Intel Core i7 (Ivy Bridge)*
  - Свако језгро процесора има
    - кеш нивоа L1, одвојено за инструкције и податке, сваки по:
      - 32 *KiB*, 512 линија по 64B
      - 8-скуп-асоцијативан
      - 4 циклуса по инструкцији
    - кеш нивоа L2
      - 256*KiB*, 512 линија по 512B
      - 8-скуп-асоцијативан
      - 10 циклуса по инструкцији
  - Процесор садржи и дељени кеш L3
    - 2-15*MiB*, величина линије 512B
    - 16-скуп-асоцијативан
    - инклузиван
    - брзина зависи од варијанте процесора

Пример савремене архитектуре
Процесор *Intel Core i7*



Пример перформанси савремене архитектуре Процесор *Intel Core i7 (2)*



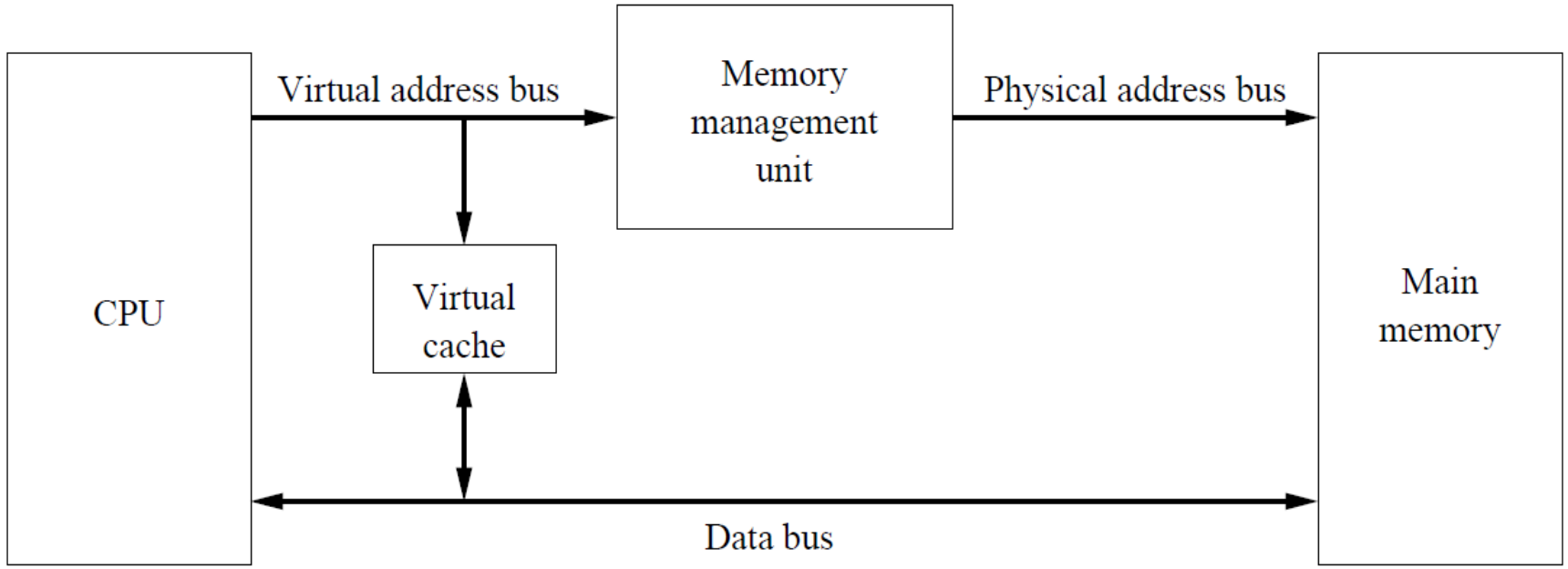
Виртуални и физички кеш

- Кеш уобичајено ради на нивоу физичких адреса
- Међутим, нема разлога да његов рад не буде на нивоу виртуалних адреса

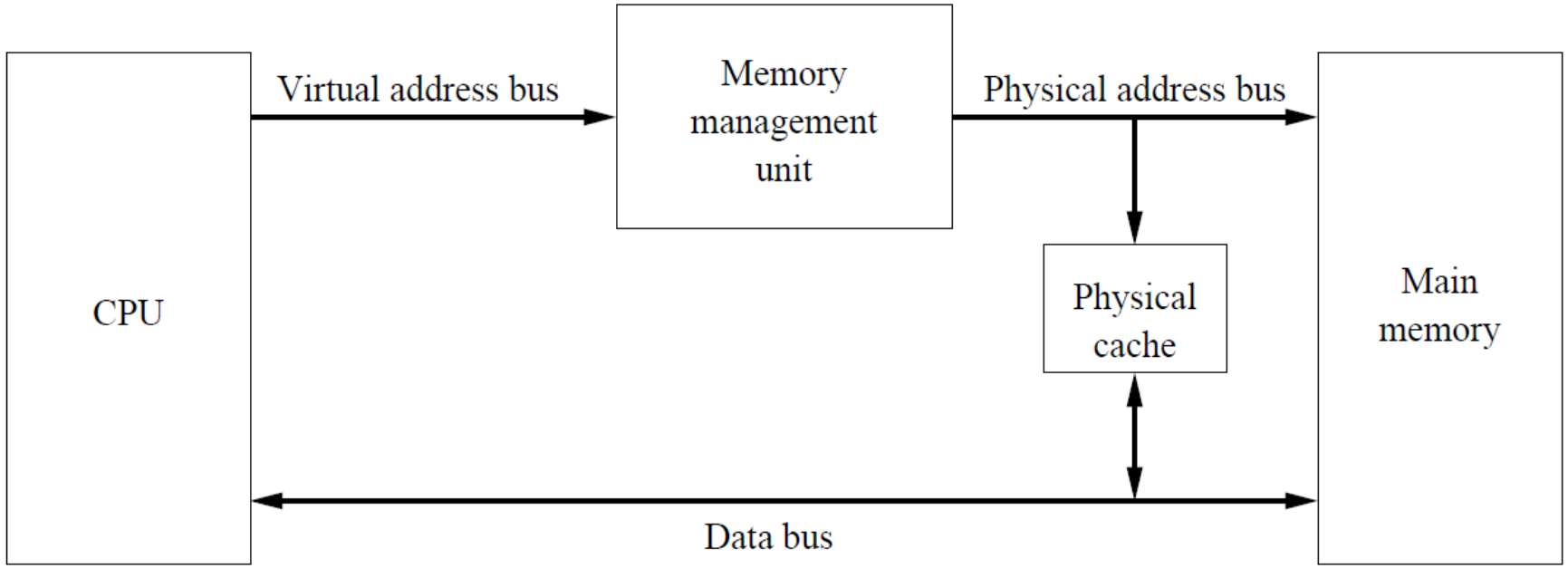
- Физички кеш се налази између транслатора адреса и меморије
- Виртуалан кеш се налази између процесора и транслатора адреса

- Кеш који је ван процесора може да буде само физички

Виртуални кеш



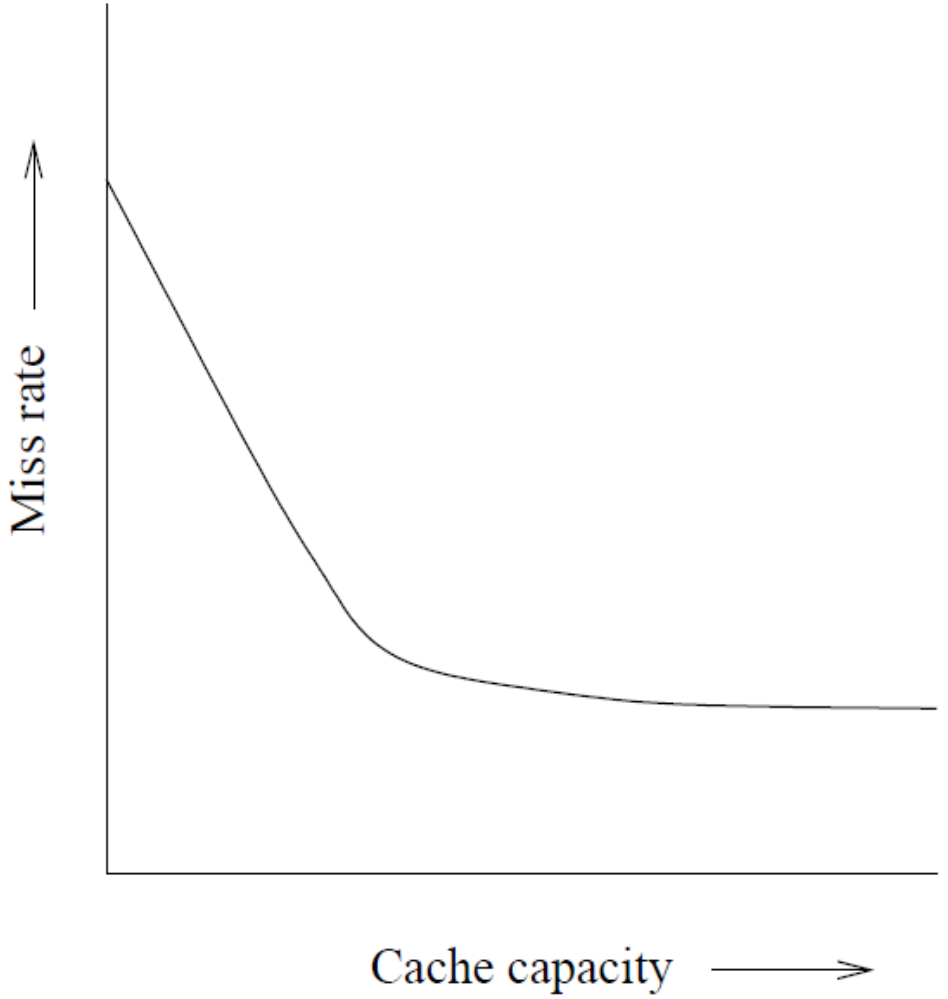
Физички кеш


CPU
Virtual address bus
Memory management unit
Physical address bus
Physical cache
Main memory
Data bus

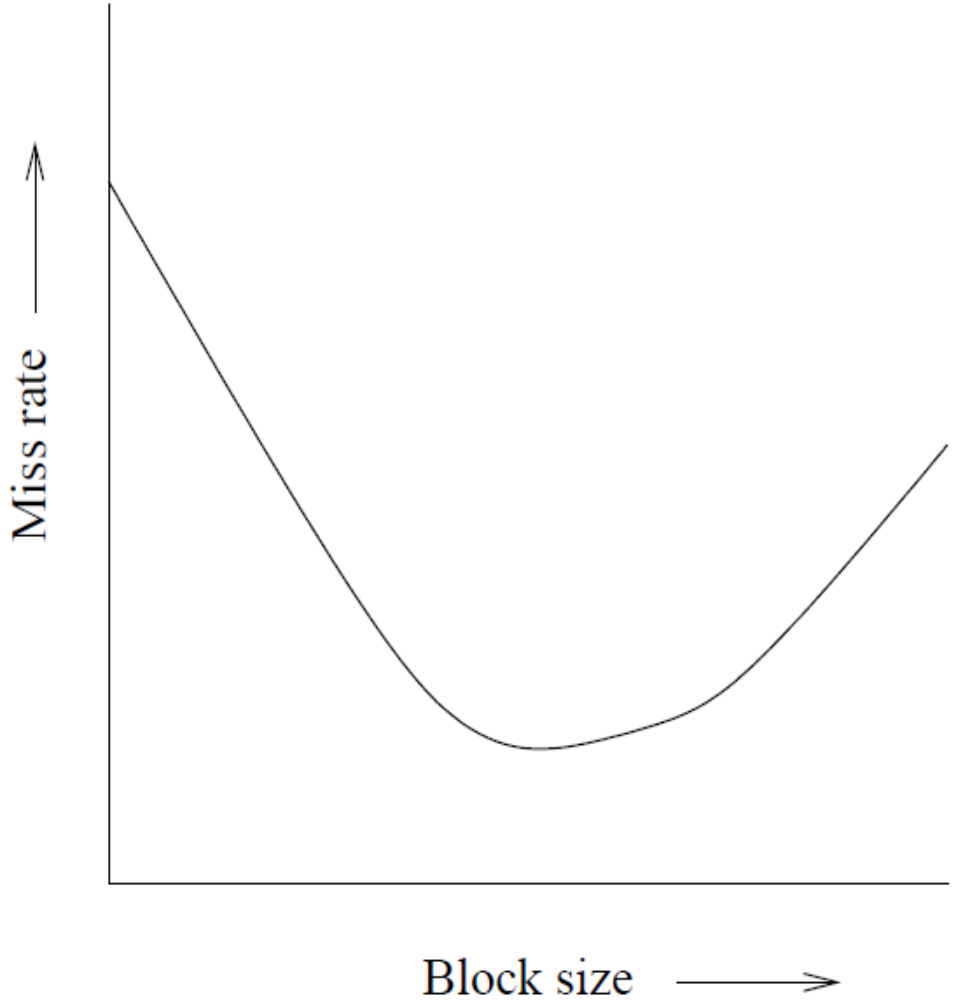
Елементи дизајна кеша

- Најважније аспекте дизаја кеша чине:
  - капацитет кеша
  - величина линија (блокова)
  - степен асоцијативности
  - јединственост или специјализованост
  - број нивоа
  - политика писања
  - виртуалан или физички кеш

Однос капацитета кеша и перформанси



Однос величине блока и перформанси (при истом капацитету)



Однос асоцијативности и перформанси (при истом капацитету и величини линија)