[P220]

Увод у архитектуру рачунара 7

Саша Малков
Универзитет у Београду
Математички факултет
2013/2014

[P271]

Увод у архитектуру рачунара

Саша Малков

Тема 8

Меморија

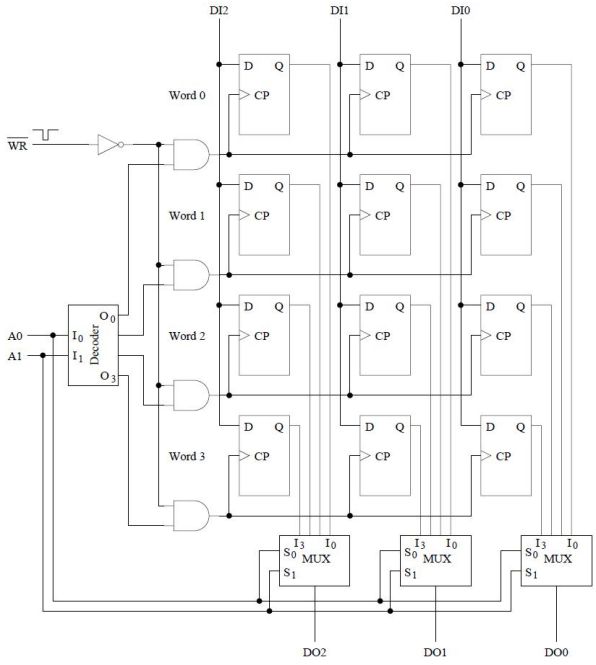
(наставак)

Меморија од *D* флип-флопова (1)

- Употребљава се дводимензиони низ *D* флип-флопова
  - сваки ред чува једну реч
  - број колона одговара броју битова у речи
    - *хоризонтално ширење* је повећавање број битова у речима
  - број редова одговара броју речи у меморији
    - *вертикално ширење* је повећавање броја речи
  - оба броја су обично неки степени броја 2
  - меморија *M* x *N* има *M* речи од по *N* битова

Универзитет у Београду - Математички факултет

Меморија имплементирана матрицом 4x3 *D* флип-флопа



Универзитет у Београду - Математички факултет

Меморија од *D* флип-флопова (2)

- Декодер одређује тачно један ред на основу улазне адресе
  - адреса је кодирана са две линије
  - декодер са И-елементима гради демултиплексор који усмерава сигнал на одговарајући ред
- Активан сигнал часовника ће добити само изабрани ред и то само у случају писања
- Сви флип-флопови у једној колони добијају исти улазни сигнал
- За читање се употребљава 4-1 мултиплексор
  - адресне линије се користе као селектори

Универзитет у Београду - Математички факултет

Ограничења и проблеми

- Није прилагођена повезивању на магистралу
  - потребно је да исте линије носе улазне и излазне податке
- Не може да се користи за прављење већих меморија
  - Потребан је додатни селекторски улаз који означава да ли се блок користи или не

Универзитет у Београду - Математички факултет

Повезивање на магистралу

- Потребно је узети у обзир
  - Исте линије се користе за улаз и за излаз
    - трансфер је двосмеран кроз исте линије
    - не могу се користити различите линије за улаз и излаз
  - Меморијска магистрала је дељена
    - само изабрани уређај сме стављати податке на магистралу података
    - остали се морају понашати као да нису повезани на магистралу
- Постоји више техника које се користе
  - Обрадићемо неке од њих

Универзитет у Београду - Математички факултет

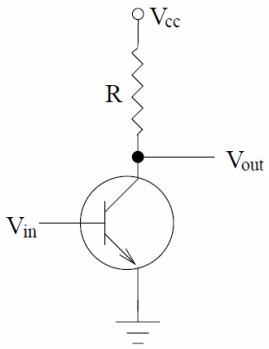
Употреба мултиплексора

- Представљен пример почива на мултиплексорима
- Мултиплексори не задовољавају наведена два захтева
  - помоћу мултиплексора не могу да се непосредно повежу улази и излази података
  - не може да се једноставно бира јединица меморије

Универзитет у Београду - Математички факултет

Употреба отворених колектора

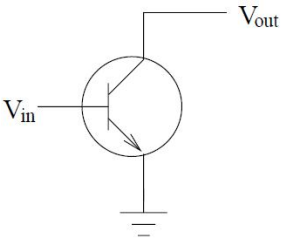
- У случају уобичајене имплементације НЕ-елемента није могуће међусобно везивање више излаза:



- Када је транзистор укључен, очекује се јачина струје *Vcc/R*
- Ако су повезана три транзистора, при чему је само један укључен (пропушта струју), он ће трпети три пута јачу струју
- У случају већег броја, јачина је сразмерна, па би транзистори прегоревали

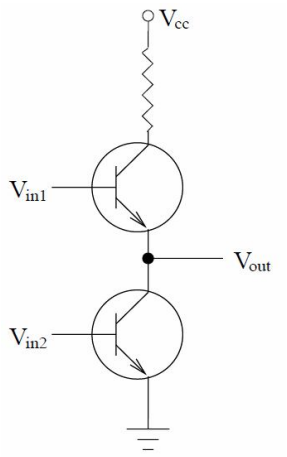
Употреба отворених колектора

- Идеја је да се уклони отпорник из кола и обезбеде отворени колекторски излази на иглицама чипа



- Отворени колектор може да има стање 0 или 1 (као и сваки излаз)
- Додатно, може да буде у стању високе импеданце (Z), када је транзистор искључен
- Уместо да се везује по отпорник и извор на сваки елемент, практично се везују једанпут за све елементе заједно

Употреба отворених колектора



- Шта би био излаз из овог кола?
- Ако су оба улаза неактивна?
  - Излаз је *одсечен, искључен*
- Ако је активан само први улаз?
  - Излаз се понаша као активан емитер = 1
- Ако је активан само други улаз?
  - Излаз се понаша као активан колектор = 0

Универзитет у Београду - Математички факултет

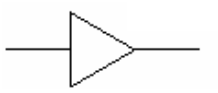
Употреба бафера са три стања

- Уређаји са три стања имају 3 а не само 2 стања (за разлику од осталих представљаних реза и флип-флопова)
  - Имају додатни контролни сигнал
  - Ако је он активан, излаз је са високом импеданцом (активан) независно од улаза
- Данас уобичајено решење

Универзитет у Београду - Математички факултет

Бафер

- Најпре да размотримо обичан “бафер”, који наизглед не служи ничему



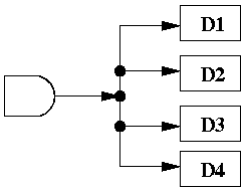
- Међутим, он има функцију
  - Бафер је *активан* елемент

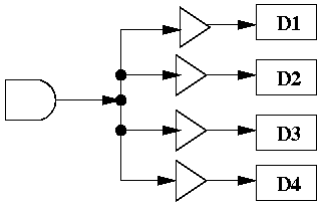
Бафер (2)

- Бафер је *активан* елемент
  - ако је на улазу *позитиван* потенцијал, он обезбеђује да је на излазу *пун* позитиван потенцијал
  - ако је на улазу *приближно нула* потенцијал, он обезбеђује да је на излазу *нула* потенцијал
- Понаша се као појачавач сигнала
  - сваки потенцијал *изнад* прага функционисања транзистора појачава се до пуног интензитета позитивног потенцијала
  - сваки потенцијал *испод* прага функционисања транзистора “појачава” се до нултог потенцијала

Бафер (3)

- У овом примеру, сваки од елемената *D1-D3* ће добити свега по ¼ потребног потенцијала
- Употреба бафера обезбеђује да елементи *D1-D3* добију пун потребан потенцијал





Универзитет у Београду - Математички факултет

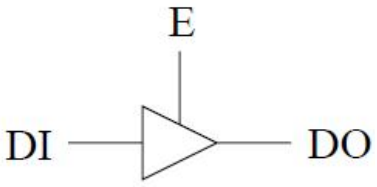
Три стања?

- До сада смо подразумевали да сваки елемент има за излаз једну од две вредности: 0 или 1
- У логичком систему то и јесте тако
- У имплементираним системима имамо потребу за трећим стањем:
  - 0 – на излазу се поставља нулти потенцијал (уземљење) и омогућава проток струје
  - 1 – на излазу се поставља позитиван потенцијал (напајање) и омогућава проток струје
  - **Z – не утиче се на стање на излазу и онемогућава се проток струје**

Универзитет у Београду - Математички факултет

Бафер са три стања

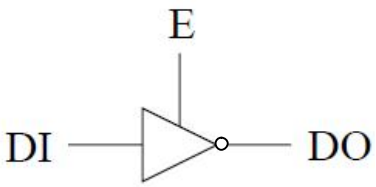
- Бафер са три стања се понаша попут **вентила**
- Ако се на "вентил" *E* који допушта проток (енгл. *enable*) доведе 0, онда се не утиче на стање на излазу
- Ако се на "вентил" *E* који допушта проток доведе 1, онда се сигнал са улаза *X* пропагира на излаз



Inputs		Output
E	DI	DO
1	0	0
1	1	1
0	X	Z

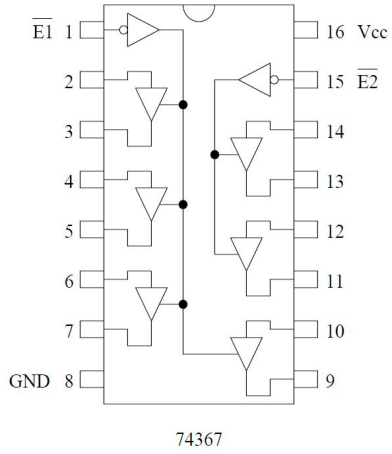
Инвертор са три стања

- Инвертор са три стања се понаша као негација са вентилом
- Ако се на "вентил" *E* који допушта проток (енгл. *enable*) доведе 0, онда се не утиче на стање на излазу
- Ако се на "вентил" *E* који допушта проток доведе 1, онда се сигнал са улаза *X* инвертује и пропагира на излаз



Inputs		Output
E	DI	DO
1	0	1
1	1	0
0	X	Z

Чип са баферима са три стања

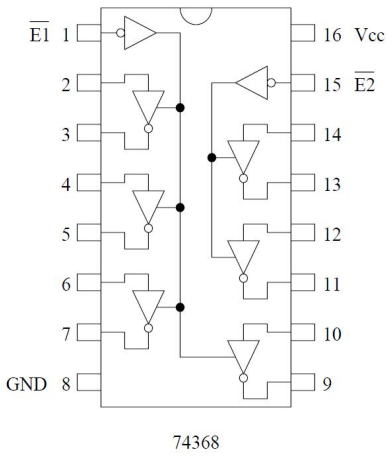


- Чип 74367
- Садржи 6 бафера са три стања
  - у две групе, од 4 и 2
- Оба прекидачка улаза су активна на ниском стању

Универзитет у Београду - Математички факултет

Чип са инверторима са три стања

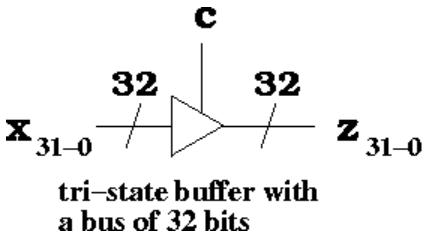
- Чип 74368
- Садржи 6 инвертора са три стања
  - у две групе, од 4 и 2
- Оба прекидачка улаза су активна на ниском стању



Универзитет у Београду - Математички факултет

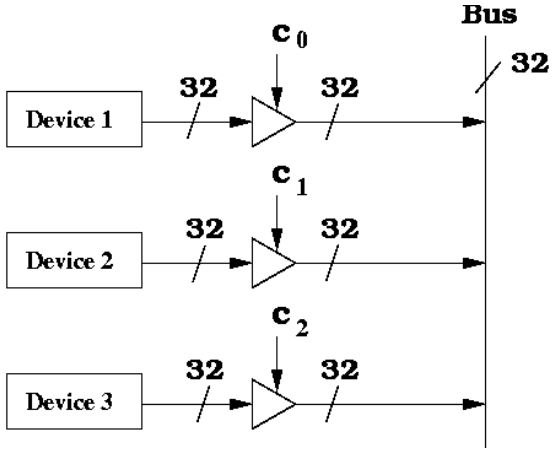
Бафери са три стања и магистрале

- На магистралу се преко бафера са три стања обично ставља истовремено већи број битова
  - у зависности од ширине магистрале
- Ради једноставнијег представљања често се користи поједностављен симбол за представљање низа бафера са три стања:



Бафери са три стања и магистрале (2)

- Пример везивања уређаја на магистралу
  - ради се о излазним подацима уређаја

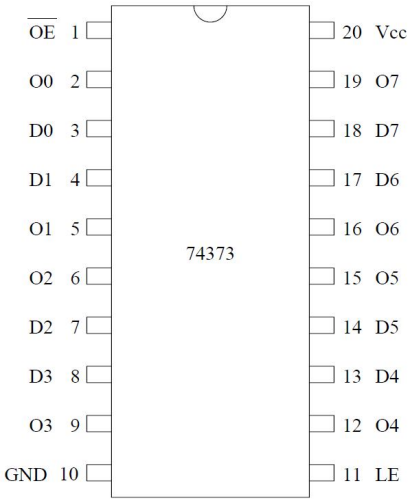


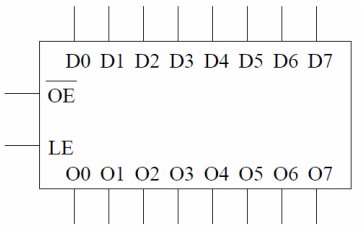
"Меморија" 74373 (1)

- Користи се као регистар ширине 8 бита
- Интерно користи 8 *D* реза
- Излазе реза шаље на излазе чипа кроз инвертујуће бафере са три стања (тј. инверторе са три стања)
  - Контролни сигнал *OE#* (*output enable*) контролише излазе
    - ако је *OE*=0 (*OE#*=1), пропушта излазе инвертора на излазе чипа
  - Контролни сигнал *LE#* контролише писање (мењање стања)
    - ако је *LE*=1 (*LE#*=0), омогућава писање
  - Због употребе резе, излаз је једнак улазу све док је *LE*=1 и *OE*=0

Универзитет у Београду - Математички факултет

"Меморија" 74373 (2)





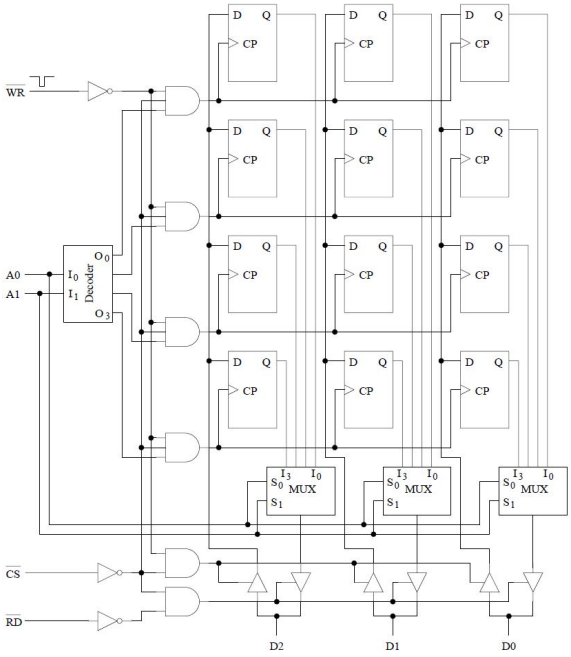
Универзитет у Београду - Математички факултет

Меморијски блок

- Потребне промене у односу на претходни дизајн:
  - Додајемо селекторски улаз
    - улазни сигнал који одређује да ли се меморијски блок употребљава или не
    - повезује се као улаз на конјункције за избор адресе
  - Спајамо улазне и излазне сигнале података
    - ако се пише, онда помоћу бафера са три стања усмеравамо податке са магистрале на улазе флип-флопова
    - ако се чита, онда помоћу бафера са три стања усмеравамо излазе из флип-флопова на магистралу

Универзитет у Београду - Математички факултет

Меморија имплементирана матрицом 4x3 *D* флип-флопа, са употребом бафера са три стања



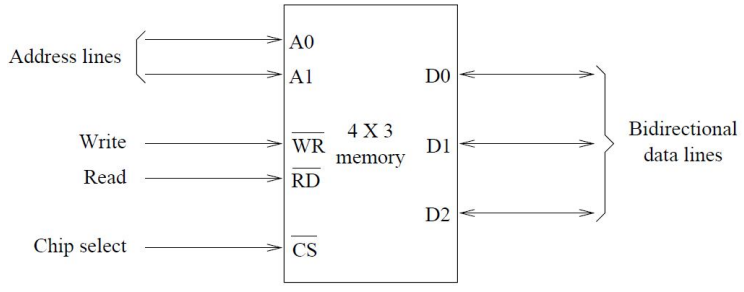
Универзитет у Београду - Математички факултет

Меморијски блок (2)

- Потребан је додатни контролни сигнал који означава операцију читања
- Селекторски улаз укључује/искључује контролне сигнале за читање и писање
- Баферима са три стања се
  - сигнал са магистрале података пропушта до улаза на флип-флопове, акко су активни и селекторски сигнал и контролни сигнал операције читања
  - сигнал са излаза флип-флопова се пропушта на магистралу података, акко су активни и селекторски сигнал и контролни сигнал операције писања
  - искључени бафер (контролни сигнал 0) има високу импеданцу и не представља сметњу функционисању укључених бафера са истим излазом/улазом

Универзитет у Београду - Математички факултет

Блок дијаграм меморије 4x3



Универзитет у Београду - Математички факултет

Прављење већих меморија

- Од меморијских блокова који имају контролне селекторе (*CS*) могу се правити већи меморијски блокови
- Први корак је прављење независне меморијске јединице која није чврсто везана за специфичне адресе у адресном простору
  - Нешто као већа верзија претходно представљеног меморијског блока
- Други корак је везивање оваквих независних меморијских јединица за конкретан адресни простор

Употреба чипа 74373 (1)

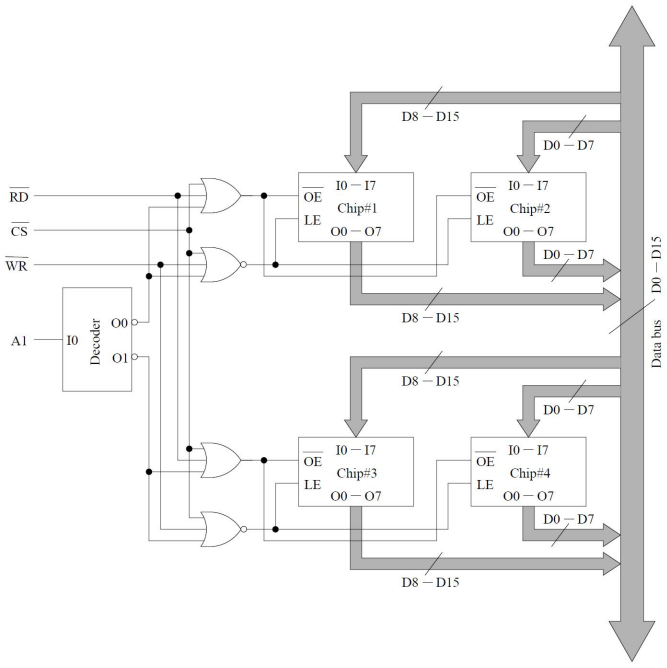
- Помоћу 4 чипа 74373 може се направити меморијски блок 2x16
  - један чип може да чува 8 бита, па се користи матрица од 2x2 чипа
- Повезивањем више чипова повећава се ширина меморијске речи (*хоризонтална експанзија*)
  - контролни улази два чипа се везују заједно како би представљали 16-битну целину
  - улазе и излазе сваког чипа везујемо на одговарајуће линије магистрале података
  - на сличан начин се може добити и шира реч
    - нпр. од 8 чипова се може добити 64-битна реч

Употреба чипа 74373 (2)

- Додавањем редова повећавамо величину меморије (*вертикална експанзија*)
  - сваки ред чува по једну реч
  - помоћу декодера се врши одабир активног реда
  - на контролне сигнале излаза чипова (*OE*) се везује конјункција
    - контролног селектора
    - контролног сигнала читања
    - излаза декодера
    - због инвертованог улаза (*OE#*) уместо конјункције се примењује дисјункција инвертованих улаза
  - на контролне сигнале улаза чипова (*LE*) се везује конјункција
    - контролног селектора
    - контролног сигнала писања
    - излаза декодера
    - због већ инвертованих аргумената, уместо конјункције се примењује дисјункција инвертованих улаза са инвертором на излазу

Универзитет у Београду - Математички факултет

Логички дијаграм меморије 2x16 изграђене од 4 чипа 74373



Универзитет у Београду - Математички факултет

Примери меморијских чипова (1)

- Постоји велики број чипова који се употребљавају за израду већих меморија
  - примери *SRAM* и *DRAM* чипова фирме *Micron*
- *SRAM*
  - 8Mb чип, у три конфигурације:
    - 512K x 18
    - 256K x 32
    - 256K x 36
  - додатни битови служе за препознавање и отклањање грешака
  - време приступа 3.5*ns*
  - чип 512K x 18 има 19 адресних линија
  - чипови 256K x 32 и 256K x 36 имају по 18 адресних линија

Примери меморијских чипова (2)

- *DRAM*
  - синхрони *DRAM*
  - 256Mb чип, у три конфигурације:
    - 64M x 4, 26 адресних линија
    - 32M x 8, 25 адресних линија
    - 16M x 16, 24 адресне линије
  - трајање циклуса је око 7*ns*

Организација чипова

- Иста количина меморије се може различито организовати и имплементирати
  - у време уских магистрала (8-16 битова) *DRAM* се израђивао са ширином од 1 бита
    - данас то није практично због велике ширине речи
- Предност широких чипова је што их је потребно мање за веће меморије
  - *Pentium* је 32-битни процесор са 64-битном магистралом података
  - меморија 16M x 64 може се направити
    - од једног реда са 4 чипа 16M x 16
    - али не и од 8 чипова 32M x 8 (добијамо 32M x 64)
  - од уских чипова се ни не могу добити неке мање меморијске јединице

Дизајн већих меморија (1)

- На примеру *DRAM*-а
- Основно питање је да ли је меморијски адресни простор (*memory address space – MAS*) адресибилан на нивоу појединачних бајтова или не
  - дужина адресибилне речи већине савремених процесора јесте 1 бајт

Дизајн већих меморија (2)

- Затим се одлучује о конфигурацији чипова
  - при изабраној циљној величини и величини чипова, не мења се укупан број чипова, али се мења њихов распоред
  - ако је циљ меморија M x N, а користе се чипови D x W:
    - број колона је N/W
    - број редова је M/D
    - број чипова је $(M \text{ x } N)/(D \text{ x } W)$
- У примеру
  - правимо меморију од $256Mib$
  - циљна конфигурација је $64Mi$ x $32b$
  - користимо чипове $16Mi$ x $16b$
  - матрица чипова је 4 x 2

Универзитет у Београду - Математички факултет

Дизајн већих меморија (3)

- Везивање на магистралу података је непосредно
  - сваком чипу у реду одговара део линија података
- Ако се у једном кораку чита N битова (>8)
  - Z најнижих битови адресе се игноришу
  - $Z = \log_2(N/8)$
- У примеру
  - 256M се адресира са 28 битова адресе
  - један чип има 24 бита адресе
  - најнижа 2 бита адресе A0, A1 се не користе
    - ширина меморије је 32 бита, а адресирање по бајтовима
  - на чипове се везују 24 бита A2 до A25
  - битови адресе A26 и A27 се користе за бирање реда чипова

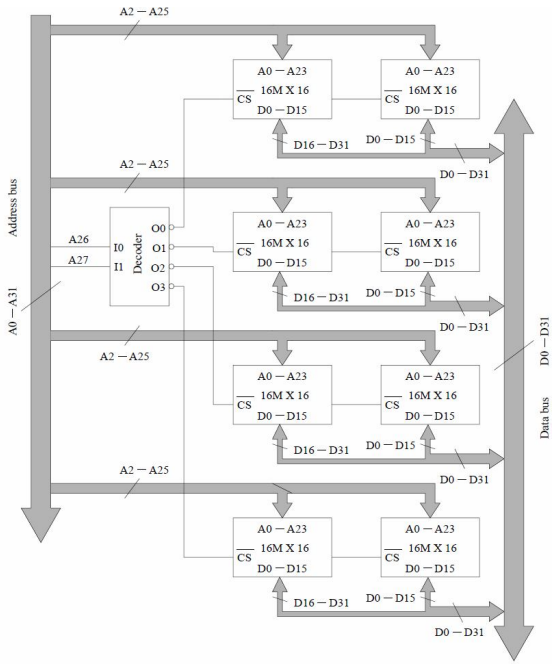
Универзитет у Београду - Математички факултет

Дизајн већих меморија (4)

- Контролни сигнали за читање и писање свих чипова се повезују међусобно и на контролну магистралу
  - (изостављено из наредног дијаграма ради једноставности)

Универзитет у Београду - Математички факултет

Дизајн меморије 64M x 32 од чипова 16M x 16



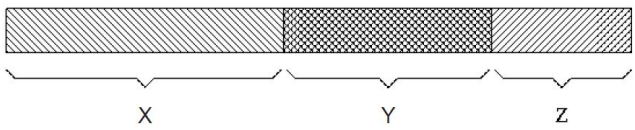
Универзитет у Београду - Математички факултет

Пресликавање адреса

- Пресликавање адреса је поступак којим се физичка меморија лоцира у адресном простору рачунара
- На пример:
  - процесор *Pentium* има адресни простор величине 4*GiB*
    - 32 адресне линије
  - ако рачунар има 128*MiB* меморије, она се може пресликати у различите адресибилне области
- Пресликавање може бити
  - пуно и
  - делимично

Врсте адресних линија

- Адресне линије се деле у три групе:
  - *X* – највише адресне линије које одређују чип
  - *Y* –адресне линије које се прослеђују чиповима
  - *Z* – најниже адресне линије које се занемарују



Пуно пресликавање адреса

- *Пуно пресликавање адреса* је оно пресликавање ког кога је функција пресликавања меморијских адреса у меморијске локације 1-1
  - за сваку меморијску локацију постоји највише једна адреса која јој одговара
- Све адресне линије X се користе при декодирању ради добијања сигнала за избор модула
- Све адресне линије се деле у две групе:
  - линије Y и Z одређују бајт у меморијском модулу
  - линије X се користе за израчунавање сигнала за избор чипа CS (*chip selector*)

Универзитет у Београду - Математички факултет

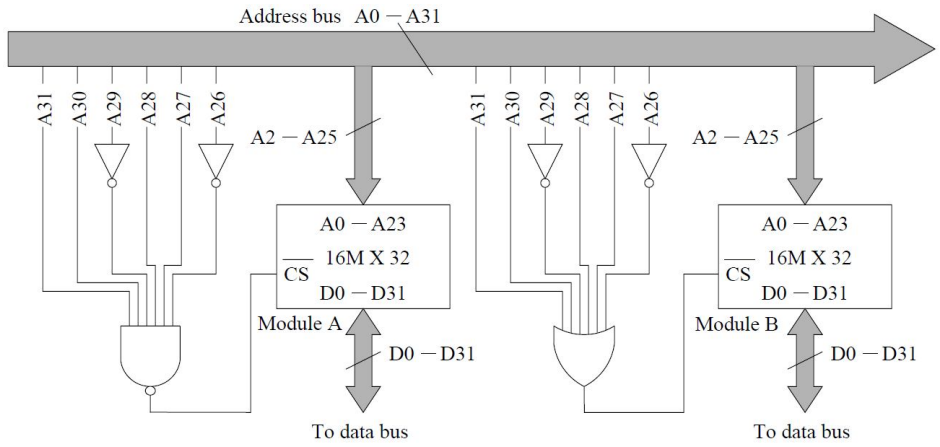
Пример пуног пресликавања адреса

- Нпр., користимо 2 модула $16Mi$ x 32 у адресном простору од $4GiB$
- Делимо 32 адресне линије на две групе:
  - нижих 26 линија (Y, Z) се користе за одређивање бајта у оквиру модула $16Mi$ x $32b$
  - виших 6 линија (X) се користе за одређивање сигнала CS
    - модулу A одговарају (на пример) адресе X = 110110
      - опсег адреса: D8000000H – DBFFFFFFH
    - модулу B одговарају (на пример) адресе X = 001001
      - опсег адреса: 24000000H – 27FFFFFFH
    - остале вредности адресе X не одговарају ниједном модулу
    - опсези се не преклапају, па је ово пуно пресликавање

(не представљамо контролне линије, ради прегледности)

Универзитет у Београду - Математички факултет

Пример пуног пресликавања адреса



Делимично пресликавање адреса

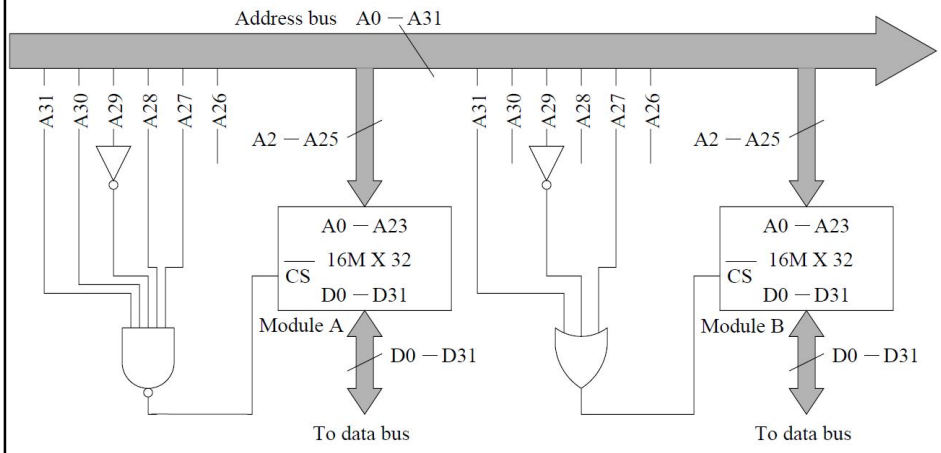
- *Делимично пресликавање адреса* је оно ког кога функција пресликавања меморијских адреса у меморијске локације није 1-1
  - неким меморијским локацијама може да одговара више адреса
- Циљ је поједностављивање логике одређивања селекторског сигнала
- Може се примењивати када је број меморијских локација значајно мањи од броја адресибилних локација

Пример делимичног пресл. адреса

- Слично претходном примеру
- Модулима се додељују вишеструке адресе
  - модулу A одговарају (на пример) адресе $X = 110110$ и $X = 110111$
    - скраћено $X = 11011d$
    - двоструки опсег адреса:
      - D8000000H – DBFFFFFFH
      - DC000000H – DFFFFFFFH
  - модулу B одговарају (на пример) адресе $X = 1d0d1d$
  - опсези се преклапају, па је ово делимично пресликавање

Универзитет у Београду - Математички факултет

Пример делимичног пресликавања адреса



Универзитет у Београду - Математички факултет

Поравнавање података

- Ако имплементација меморије почива на 32-битним модулима, онда
  - ако је адреса поравната са 32-битним речима, читање 32-битне речи се одвија у једном циклусу
    - модул може вратити целу 32-битну реч одједанпут
  - ако адреса није поравната са 32-битним речима, читање 32-битне речи се одвија у два циклуса
    - модул не може вратити целу 32-битну реч одједанпут

Универзитет у Београду - Математички факултет

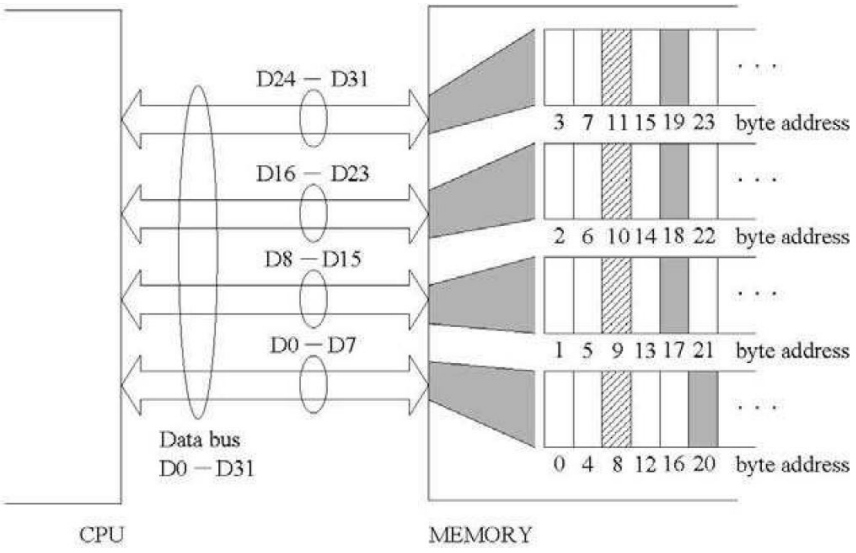
Поравнавање података (примери)

- У случају фамилије процесора *Intel x86*
  - могу се задавати непоравнате адресе
  - поравнатост има утицаја на перформансе
  - тзв. *услов мекоī ūоравнања*
- У случају фамилија *Motorola 68000, Intel i860*
  - не смеју се задавати непоравнате адресе
  - тзв. *услов ŧврдоī ūоравнања*

Универзитет у Београду - Математички факултет

Пример поравнатих и непоравнатих адреса

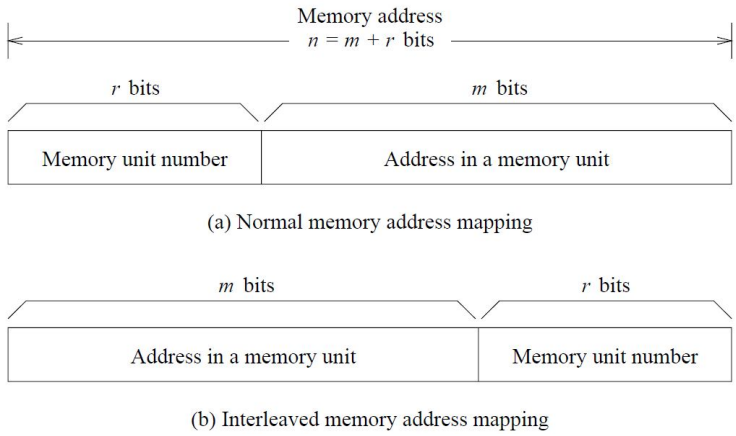
- Адреса 8 је поравната
  - 32-битни податак се чита у једном циклусу
- Адреса 17 није поравната
  - 32-битни податак се чита у два циклуса



Испреплетане меморије (1)

- Као што смо видели, уобичајено је да се виших r адресних линија употребљавају за препознавање модула, а нижих m за адресирање у модулу
- Код испреплетаних меморија то се мења, како би се узастопне речи налазиле у различитим модулима

Испреплетане меморије (2)



Испреплетане меморије (3)

- Раније представљен дизајн меморије омогућава њено једноставно повећавање
- Сваки захтев се извршава током више циклуса (нпр. 4)
- Ако бисмо желели да прочитамо 8 узастопних речи, то би захтевало 8 x 4 = 32 циклуса
- Испреплетане меморије омогућавају да се скрати време читања узастопних речи

Испреплетане меморије (4)

- Меморијски модули се у контексту преплитања називају *меморијским банкама*
- Адресе се додељују банкама наизменично:
  - нека имамо *B* банака
  - банка се одређује као *addr MOD B*
- Имплементирају се на два начина:
  - синхронизованим приступом и
  - независним приступом

Универзитет у Београду - Математички факултет

Синхронизован приступ

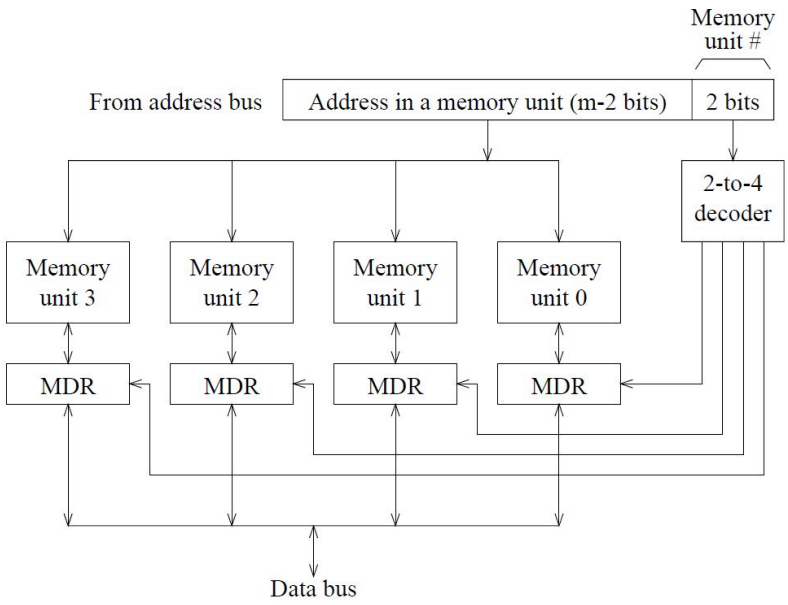
- Горњих *r* адресних линија се симултано доводе свим модулима
  - Све банке истовремено започињу своје операције
  - Након 4 циклуса, свака од меморија је довршила читање
    - (претпоставимо да читање захтева 4 циклуса)
- Прочитани подаци се уписују у четири меморијска регистра (*MDR – memory data register*)
  - имплементирани помоћу бафера са три стања
- Током наредних *B* циклуса се ови подаци преносе на магистралу
  - декодером се бирају редом регистри
- Док се ови подаци преносе, наредних *B* речи се читају из меморије

Универзитет у Београду - Математички факултет

Синхронизован приступ (2)

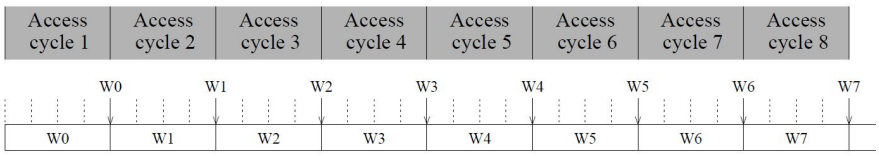
- За читање првих B речи је потребно 4 циклуса
- Свака следећа реч захтева мање

Испреплетана меморије са 4 банке – синхронизован приступ

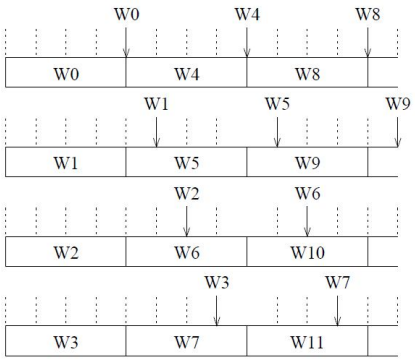


Испреплетана меморије са 4 банке – синхронизован приступ – ток читања

Без преплитања



Са преплитањем



- За читање 8 речи
- Без преплитања
  - 32 циклуса
- Са преплитањем
  - 11 циклуса

- За читање 12 речи
- Без преплитања
  - 48 циклуса
- Са преплитањем
  - 15 циклуса

Универзитет у Београду - Математички факултет

Независан приступ (1)

- Проблем са синхронизованим приступом је да доноси убрзања само у случају секвенцијалног приступа
- Независан приступ омогућава преклопљено (*pipelined*) извршавање операција на произвољним адресама
- Свакој банци се додаје меморијски регистар адресе (*MAR*) који чува адресу коју користи та банка
- Нису потребни додатни регистри података (*MDR*), већ се подаци читају непосредно из меморије

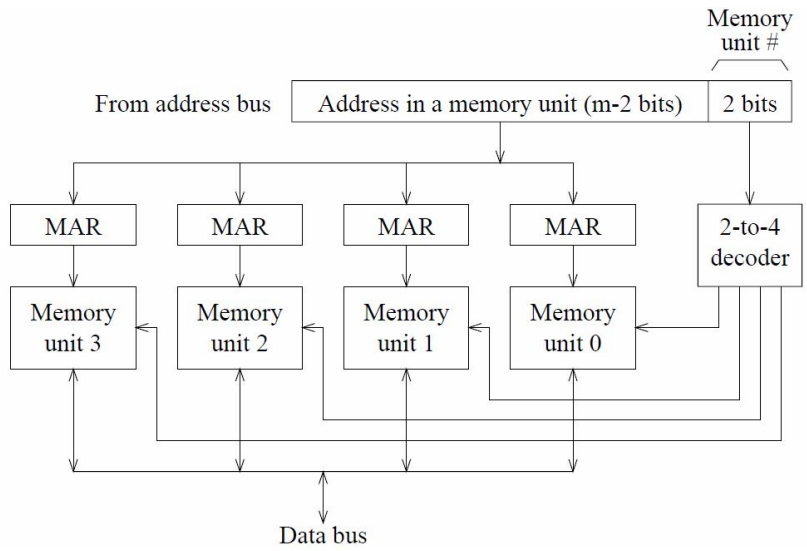
Универзитет у Београду - Математички факултет

Независан приступ (2)

- Независан приступ има исти ниво преклапања као и синхронизован
  - преклапање се постиже на нивоу адреса, а не на нивоу података
  - у сваком циклусу по једна адреса иде у одговарајући *MAR*
  - током 4 циклуса се чита податак и испоручује на магистралу

Универзитет у Београду - Математички факултет

Испреплетана меморије са 4 банке – независан приступ



Универзитет у Београду - Математички факултет

Број банака

- Број банака би требало да буде бар једнак броју циклуса
  - Због тога је претходно синхронизован приступ илустрован примером са 4 банке и 4 циклуса

Проблеми у вези преплитања

- Преплитање захтева сложенију имплементацију меморијских кола
  - додатни регистри (података или адресе)
  - додатна контролна кола за постављање података у регистре и читање из њих
  - слаба толеранција грешака
    - ако је једна банка у квару, читава меморија не ради
  - умањена флексибилност повећавања меморије

[P271]

Увод у архитектуру рачунара

Саша Малков

Тема 9

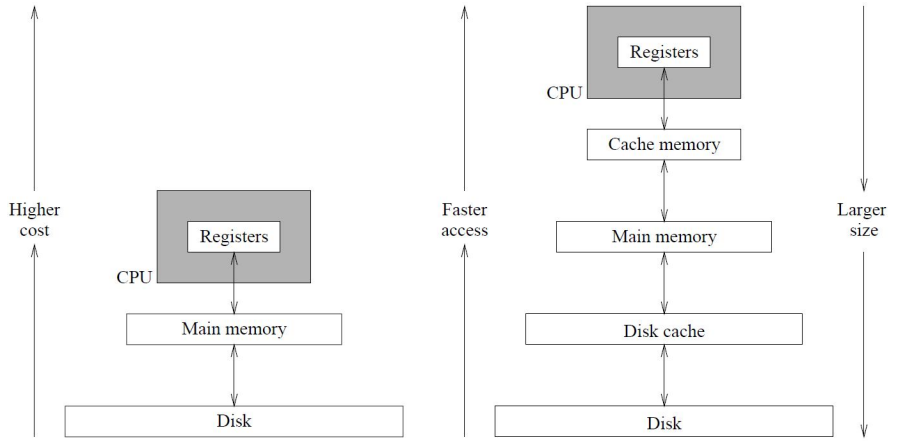
Кеш

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



Универзитет у Београду - Математички факултет

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)

Универзитет у Београду - Математички факултет

Принцип рада кеша

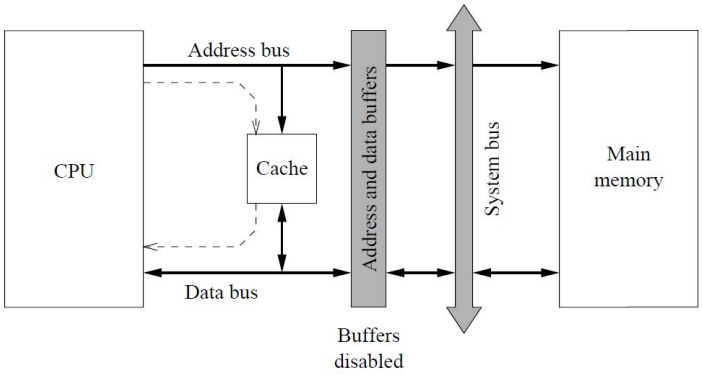
- Основни принцип рада је захватање података из радне меморије унапред (*prefetch*), пре него што заиста затребају процесору
  - ако је успешно предвиђено који ће подаци бити потребни процесору у блиској будућности, њих ће процесор читати из кеша а не из меморије
  - тиме се значајно подижу перформансе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



Универзитет у Београду - Математички факултет

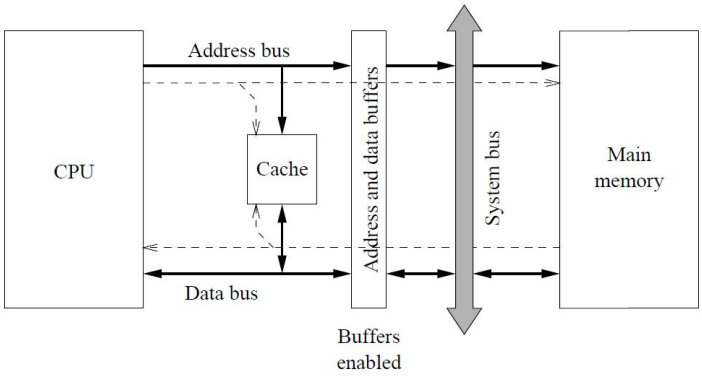
Читање у случају поготка (2)

- 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

Универзитет у Београду - Математички факултет

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)

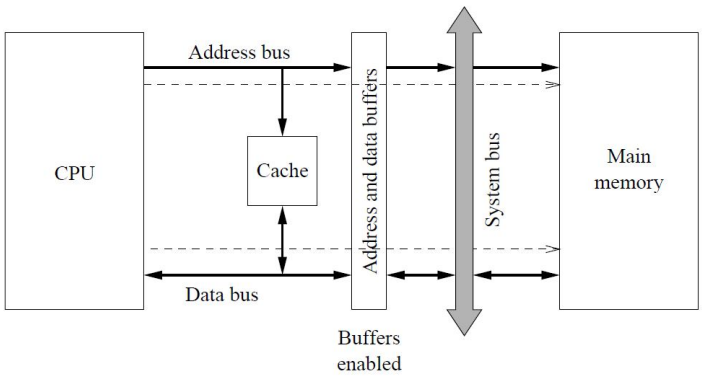
- У случају промашаја, линије адресе и података према меморији су активне
- Одвија се уобичајено (као да нема кеша) читање података из меморије
  - додатно се прочитани подаци уписују и у кеш
- Читање са промашајем је нешто спорије од читања из меморије без примене кеша
  - због неопходног проверавања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

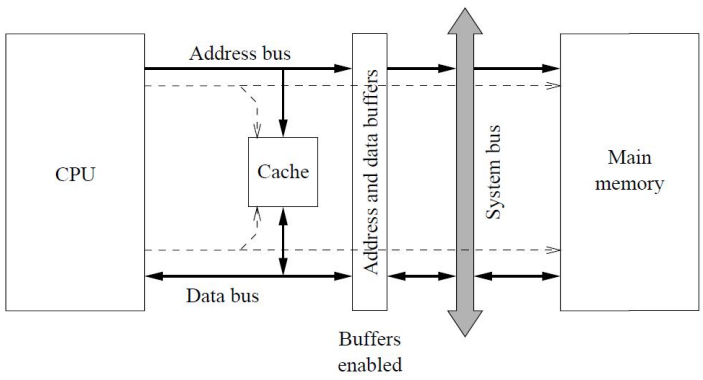


Писање у случају поготка

- 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?

Универзитет у Београду - Математички факултет

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;
```

Универзитет у Београду - Математички факултет

Зашто кеш ради? (2)

- Први фактор за успешну примену кеша је *понављена 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

Универзитет у Београду - Математички факултет

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

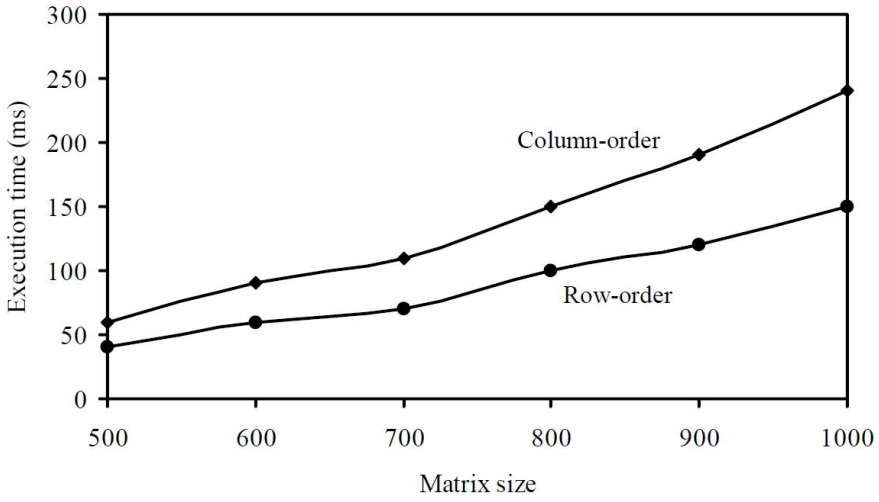
Универзитет у Београду - Математички факултет

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

Универзитет у Београду - Математички факултет

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

Универзитет у Београду - Математички факултет

Литература

- *Sivarama Dandamudi*, ***Fundamentals of Computer Organization and Design***, *Springer, 2002.*
- *Ненад Митић*, ***Основи рачунарских система***, *Математички факултет, 2002.*