Факултет инжењерских наука Универзитета у Крагујевцу

Назив студијског програма: Машинско инжењерство
Ниво студија: Основне академске студије
Модул: Информатика у инжењерству
Предмет: Архитектура рачунарских система
Број индекса: 719/2013

Лазар Р. Максимовић

Перформансе рачунарских система

завршни рад

Комисија за преглед и одбрану:

1. _____
2. _____
3. _____

Датум одбране:_____

Оцена:_____

Задатак рада

Перформансе су кључни критеријум код пројектовања, набавке, и коришћења рачунарских система. Циљ купца и корисника је да добију систем са најбољим перформансама по најнижој цени. Да би остварили овај циљ, пројектанти треба добро да познају методологије и технике на основу којих се врши процена перформанси раунарских система, а корисници да поседују основна знања о коришћењу терминологије и значењу тих перформансних метрика.

Када пројектанти саопште резултате о евалуацији, купци треба добро да разумеју на шта се те перформансе односе и шта оне конкретно значе. У раду се описују и анализирају мере за оцену перформанси рачунара: време одзива (корисничко процесорско време, време приступа меморији, време приступа дисковима, време потребно за обављање U/I активности, додатно време због позива ОС), процесорско време, MIPS (Милион инструкција по секунди), MFLOPS (Million Floating-Point Operations Per Second), benchmark програми, убрзање итд.

Препоручена литература:

1. William Stallings, *Организација и архитектура рачунара*, Пројекат у функцији перформанси, 2006.
2. Andrew S. Tanenbaum, *Архитектура и организација рачунара*, 2005.
3. Слободан Обрадовић, *Рачунари*, 2003.

Крагујевац, 10.03.2020.

Ментор:
Проф. Др. Јасна Радуловић

_____

Резиме:

У раду су описане перформансе процесора, дати су примери за побољшање перформанси процесора. Приказани су језици високог ниво који служе за бржи рад процесора а самим тим утичу и на његове перформансе. Приказане су перформансе меморијског система рачунара (кеш меморија, оперативне меморије). Описан је рад и перформансе спољне меморије (магнетни диск), као и заштита података на спољној меморији RAID.

Кључне речи: CPU Time, RAM, CPI, pipelining, *MIPS,* кеш меморија, магнетни диск, RAID.

Abstract:

The paper describes the performance of the processor, gives examples for improving the performance of the processor. High-level languages are presented, which serve for faster processor operation and thus affect its performance. The performance of the computer's memory system (cache memory, operative memory) is shown. The operation and performance of external memory (magnetic disk) are described, as well as data protection on external RAID memory.

Key words: CPU Time, RAM, CPI, pipelining, *MIPS,* cache memory, magnetic disk, RAID.

Садржај:

Увод

Еволуцију рачунара су обележили повећање брзина процесора, смањивање величине рачунара, повећавање величине меморије и повећање капацитета и брзине U/I.

Један од чинилаца одговорних за велико повећање брзине процесора јесте смањење величине микропроцесорских компонената; то смањује растојање између компонената и стога повећава брзину. Међутим, прави добитак у брзини последњих година последица је организације процесора, укључујући обимну употребу техника проточне обраде и паралелног извршења и коришћења технике спекулативног извршења, која резултује пробним извршењем будућих инструкција које би могле бити потребне. Све те технике су пројектоване да би се процесор држао запосленим у што је могуће већем делу времена.

Најважније питање у пројектовању рачунара јесте уравнотежење перформансе разних елемената, тако да добици у перформанси у једној области не буду ометени заостајањем у другим областима. Посебно, брзина процесора се више повећавала од времена приступа меморије. Разне технике се користе да би се компензовала та несагласност, укључујући кеш меморије, шире путање података од меморије до процесора и интелигентније меморијске чипове.

У раду су описане перформансе процесора као најбитније компоненте рачунарског система:

- Приказано је процесорско време CPU time, време које процесор проводи на извршењу одређеног програма.
- Описано је извршавање инструкција код старијих и код новијих процесора.
- Приказани су параметри за дефинисање перформанси једног процесора који представљају брзину извршавања програма.

Меморијски подсистем рачунара служи за складиштење бинарних информација и састоји се од низа меморијских компоненти, које се међусобно разликују по капацитету и брзини времена приступа.

Кроз овај рад описане су перформансе меморисног система рачунара:

- Описане су перформансе кеш меморије која се налази између процесора и главне меморије и има улогу да премости јаз који се јавља услед велике брзине процесора и меморије.
- Оперативна меморија представља скуп интегрисаних кола која служе за памћење бинарних података. Два основна облика полупроводничке меморије са случајним приступом динамичка DRAM и статичка SRAM меморија.
- Спољна меморија представља меморију намењену за трајно складиштење података. Као најважнија компонета спољне меморије је магнетни диск.

Да би се постигла боља перформанса и већа расположивост, сервери и већи системи користе RAID технологију дискова.

1. ПЕРФОРМАНСЕ ПРОЦЕСОРА

Најједноставнији параметар којим се могу мерити перформансе рачунарског система је време извршавања одређеног програма енг. *elapsed time*. Перформансе рачунарског система су обрнуто пропорционалне времену извршавања програма; краће време извршавања значи да рачунарски систем поседује боље перформансе. Поређењем времена потребног за извршавање одређеног програма на два рачунарска система, могуће је утврдити однос перформанси та два рачунарска система. Ово време представља укупно протекло време од почетка извршавања програма до његовог завршетка. Потребно је напоменути да ово време није у потпуности меродаван параметар, јер рачунарски систем поред тестираног програма извршава и низ других програма услед *timesharing*-а и *multitasking*-а, који повећавају укупно време извршења испитиваног програма. Због тога се уводи процесорско време *CPU time*, које представља време које процесор искључиво проведе на извршавању одређеног програма.

Процесорско време које представља време потребно процесору за извршавање програма може се рашчланити на више чланова према следећој формули на:

$$\text{CPU Time} = \frac{\text{time}}{\text{program}} = \frac{\text{time}}{\text{cycle}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{instructions}}{\text{program}}$$

$\frac{\text{time}}{\text{cycle}}$ → **Clock Cycle time**; $\frac{\text{cycles}}{\text{instruction}}$ → CPI; $\frac{\text{instructions}}{\text{program}}$ → Instruction Count

(1)

Као што се из приложене формуле види на брзину извршавања неког програма утичу следећа три параметра:

- Време трајања једног тактног интервала (*Clock Cycle Time*)
- Просечна дужина инструкције у циклусима(*Cycles Per Instruction - CPI*)
- Број инструкција у извршаваном програму (*Instruction Count*)

Време трајања једног тактног интервала (*Clock Cycle Time*) представља време трајања једног тактног импулса и оно је обрнуто пропорционално фреквенцији на којој ради процесор. Процесор представља секвенцијалну логичку мрежу састављену од низа дигиталних склопова, која захтева јединствени тактни сигнал који обезбеђује синхронизацију свих делова процесора. Повећање фреквенције рада процесора ограничено је технологијом израде процесора.

Током дугог низа година, радна фреквенција и број транзистора унутар процесора се повећавао, док је се смањивала резолуција технологије израде.

Последњих година заустављен тренд раста фреквенције процесора на око 4GHz. Два кључна проблема која су зауставила даље повећање брзине процесора су:

- Време кашњења у логичким колима и време пропагације сигнала између логичких кола

- Повећано загревање услед високе радне фреквенције и великог броја транзистора у процесору.

Просечна дужина инструкције у циклусима (*Cycles per instruction-CPI*) представља број тактних циклуса који су потребни да се изврши једна инструкција. У случају RISC (Reduced Instruction Set Computer) процесора све инструкције су једнаке дужине и просечна дужина инструкције је константан. У случају процесора комплексних архитектура CISC (Complex Instruction Set Computer)) инструкције су различитих дужина па се просечна дужина инструкције одређује статистички на основу заступљености одређених инструкција у програму који се извршава.[2]

Пример 1. Прорачун просечне дужине инструкције код CISC процесора

Instruction	# cycles	% Instr. Count
Load	5	25
Store	5	15
ALU	5	40
Branch/Jump	3	15
Others	6	5

$$CPI = \sum L_i \cdot p_i$$

$$CPI = L_{Load} \cdot p_{Load} + L_{Store} \cdot p_{Store} + L_{ALU} \cdot p_{ALU} + L_{Branch} \cdot p_{Branch} + L_{Others} \cdot p_{Others}$$

$$CPI = 5 \cdot 0.25 + 5 \cdot 0.15 + 5 \cdot 0.4 + 3 \cdot 0.15 + 6 \cdot 0.05 = 4{,}75 \frac{instrukcija}{taktu} \quad (2)$$

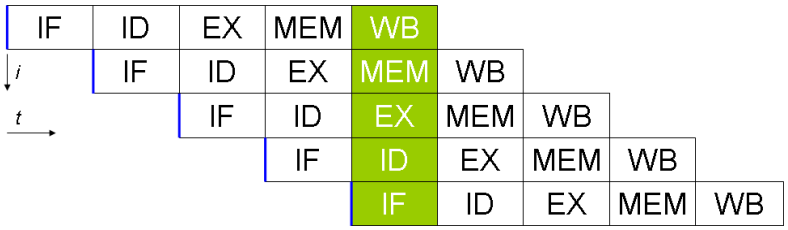
Код архитектуре старијих процесора инструкције су извршаване једна за другом, и извршавање једне инструкције, почињало је тек када се заврши претходна инструкција. Претпоставимо да се свака инструкција састоји из пет фаза:

- IF(Instruction Fetch) – дохватање инструкције,
- ID(Instruction Decode) – декодирање инструкције,
- EX(Execute) - извршавање,
- MEM(Memory access) – приступ меморији,
- WB(Register write back) – упис резултата [6]

Ако свака фаза инструкције траје један тактни интервал, онда је за извршење три сукцесивне инструкције потребно 15 тактних импулса, односно у просеку 5 тактних импулса по једној инструкцији, као што је приказано на следећој слици.

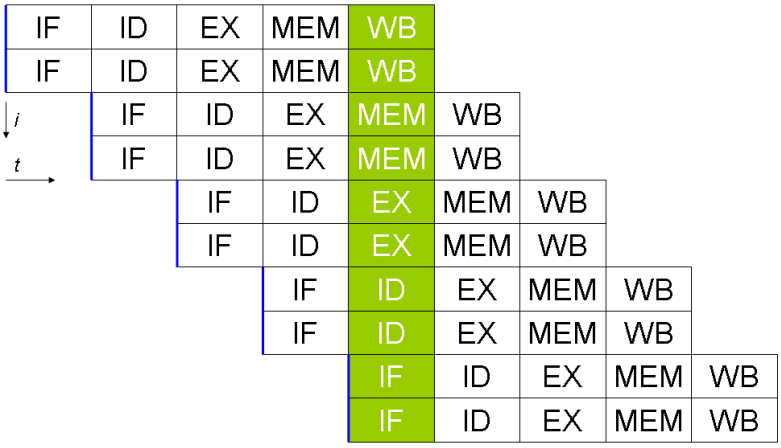
Слика 1. Рад једноставног процесора без pipelininga [7]

Пошто различити делови процесора учествују у различитим фазама извршења инструкције, примећујемо да су склопови процесора у току извршења једне инструкције већи део времена неискоришћени. Стога се у новијим архитектурама процесора примењује паралелизам на нивоу инструкција, тзв. *pipelining*. Овим приступом, пошто један део процесора заврши једну фазу једне инструкције, он затим преузима исту фазу следеће инструкције. Ако свака инструкција поседује пет фаза у току свог извршења, то значи да се у једном тактном интервалу може истовремено извршавати пет фаза различитих инструкција. Стога ће извршавање пет инструкција на оваквом рачунару трајати девет тактних интервала, односно 1.8 тактних интервала по једној инструкцији, што је у многоме брже од процесора који не поседује *pipelining*. [2]



Слика 2. Рад процесора са петостепеним pipelining -ом [8]

У последње време почело се са применом суперскаларних архитектура процесора, који поседују две или више процесних јединица које у исто време могу извршавати две или више инструкција. На слици је приказан ток извршења инструкција на дво-језгарном процесору који поседује петостепени *pipelining*, при чему се у току девет тактних интервала изврши десет инструкција, односно 0.9 тактних интервала по једној инструкцији. [6]



Слика 3. Рад двојезгарног процесора са петостепеним pipelining -ом [9]

Перформансе процесора на нивоу архитектуре рачунара могуће је побољшати проширењем опсега података над којим се могу извршавати инструкције. Првобитни процесори персоналних рачунара су извршавали инструкције над осмобитним подацима, док савремени процесори извршавају инструкције над 64 битним подацима.

Број инструкција у извршаваном програму (*Instruction Count*) представља број машинских инструкција од којих се састоји написани програм. Он у многоме зависи од сета расположивих машинских инструкција којима располаже посматрани процесор. Један од начина да се смањи дужина извршавања појединих операција је коришћењем хардверски подржаних инструкција. Тако савремени процесори поседују аритметичке јединице за рад са целим бројевима, аритметичке јединице за рад у покретном зарезу и низ других јединица које омогућавају извршавање комплексних операција за време једне инструкције од пар тактних интервала. Тиме се значајно штеди на времену извршавања али за последицу има значајно повећање комплексности процесора. Уколико процесор не поседује инструкције којима се извршавају комплексне инструкције, оне се могу извршити софтверски помоћу мање комплексних инструкција. Овакав приступ за последицу има значајно повећање времена извршења операција јер се операције извршавају помоћу више десетина или стотина једноставних инструкција.

У следећем примеру приказана је разлика у времену извршења одређених операција множења целих бројева различитих типова у случају коришћења хардверски подржане инструкције осмобитног неозначеног множења и софтверским извршењем операција коришћењем методе узастопног сабирања.

Пример 2. Разлика између хардверски и софтверски подржаних операција

Табела 1. Поређење перформанси за различите операције множења

Routine	Multiply Method	Program Memory (Words)	Cycles (Max)	Time		
				@ 40 MHz	@ 10 MHz	@ 4 MHz
8 x 8 unsigned	Without hardware multiply	13	69	6.9 µs	27.6 µs	69 µs
	Hardware multiply	1	1	100 ns	400 ns	1 µs
8 x 8 signed	Without hardware multiply	33	91	9.1 µs	36.4 µs	91 µs
	Hardware multiply	6	6	600 ns	2.4 µs	6 µs
16 x 16 unsigned	Without hardware multiply	21	242	24.2 µs	96.8 µs	242 µs
	Hardware multiply	28	28	2.8 µs	11.2 µs	28 µs
16 x 16 signed	Without hardware multiply	52	254	25.4 µs	102.6 µs	254 µs
	Hardware multiply	35	40	4.0 µs	16.0 µs	40 µs

Највећи део програма се пише у језицима високог нивоа при чему се овако написан код преводи на машински језик коришћењем програмског преводиоца. Програмски језици високог нивоа у зависности од процеса превођења могу се поделити на компајлерске и интерпретерске језике. Компајлерски језици преводе код написан као целину, при чему је циљ да се произведе оптимални машински код. У зависности од тренутка превођења разликујемо: [2]

- статичко комплајлирање - програме се преводе у машински код приликом креирања програма и потом се сваки пут покреће преведени машински код - овако написани програми се брзо извршавају јер не губи процесорско време на превођење приликом извршавања
- динамичко компајлирање - програми се преводе непосредно пре извршавања програма - овако написани програми постижу нешто лошије перформансе јер се губи процесорско време на превођење приликом извршавања.

Интерпретерски језици преводе сваку наредбу кода високог нивоа замењују одговарајућим већ преведеним деловима кода који се састављају машински код. Услед оваквог начина превођења, постиже се знатно лошија оптимизација кода у односу на компајлерске језике, што за последицу има лошије перформансе написаних програма.

Смањење броја инструкција написаног у језику високог нивоа могуће је постићи и оптимизацијом процеса компајлирања. Оптимизацијом је могуће да компајлер изврши измене у коду виског нивоа како би се преведени програм био краћи и брже се извршавао. Уколико програмер декларише промењљиве или функције које се касније у програм не користе, компајлер их може одстранити у процесу компајлирања. Као резултат различитих техника оптимизације процеса компајлирања могуће је добити два типа оптимизованог програма:

- Програм код кога је извршена оптимизација величина кода - код оваквог типа оптимизација користе се позиви функција, петља и друге програмске структуре како би се смањила величина кода док време извршавања није приоритетан параметар
- Програм код кога је извршена оптимизација времена извршавања - код оваквог типа оптимизација користе се технице угнежђивања тела функција у основни код (Code inlinig), технике развијања петљи (Loop unrolling) и друго, како би се смањило време извршавања док се повећава величина кода јер она није приоритетан параметар. У следећем примеру је приказано угнежђивање теле функције.

Пример 3. Ако је дефинисана функција за размену вредности две промењљиве:

```
void swap(int & m, int & n)
{
  int temp = m;
  m = n;
  n = temp;
}
```

У главном делу програма функција је прозвана са следећим позивом:

```
swap(x, y);
```

Након оптимизације кода у главном делу програма уместо позива функције биће угнежђено тело функције, тако да се неће губити време на прекид главном програма и прелазака на извршење функције. Уколико се функција често позива у коду, њено тело ће

бити уметнуто на сваком месту где се она позива што знатно може утицати на величину програма.

```
int temp = x;
x = y;
y = temp;
```

У примеру четири можемо видети развијање петље са фиксним и познатим бројем интерација.

Пример 4. Развијање петље. Ако је дефинисана петља за сабирање два низа са по 10 елемената:

```
for(int i=0;i<10;i++)
        a[i] = b[i] + c[i];
```

Иако петља има фиксан и познат број итерација у свакој од итерација ће се проверавати да ли је то последња итерација чиме ће се губити део процесорског времена. Како би се број провера смањио, може се повећати корак итерација на две итерације по петљи.

```
for(int i=0;i<10;i+=2)
        {
        a[i] = b[i] + c[i];
        a[i+1] = b[i+1] + c[i+1];
        }
```

Пошто је број итерација у петљи статичан, она се у потпуности може уклонити:

```
a[0] = b[0] + c[0];
a[1] = b[1] + c[1];
a[2] = b[2] + c[2];
a[3] = b[3] + c[3];
a[4] = b[4] + c[4];
a[5] = b[5] + c[5];
a[6] = b[6] + c[6];
a[7] = b[7] + c[7];
a[8] = b[8] + c[8];
a[9] = b[9] + c[9];
```

Ефикасност кода написаног у језику високог нивоа не зависи само од квалитета програмског преводиоца већ зависи и од вештина програмера у ефикасном коришћењу променљивих, програмских структура и оптималних алгоритама.

У примеру пет су приказани алгоритми за сортирање Bubble sort, Quick sort, Матлаб.

Пример 5. Коришћење оптималног алгоритма за сортирање**.** Bubble sort је мање ефикасан у односу на Quick sort алгоритам, док је Матлабов алгоритам поседује најбоље перформансе јер је заснован на принципу разгранатог стабла.

```
n=10000; % Broj clanova u nizu
a=round(1000*rand(n,1)); % Generisanje niza sa nasimučnim članiovima
b=round(1000*rand(n,1)); % Generisanje niza sa nasimučnim članiovima
c=round(1000*rand(n,1)); % Generisanje niza sa nasimučnim članiovima

fprintf (1, 'Sortitanje niza koriscenjem Bubble sort algoritma \n')
sorted = false;
tic; %Pokreni štopericu
while sorted ~= true
   sorted = true;
   n = n -1;
for i=1:m
if a(i,1) > a(i+1,1)
   temp = a(i,1);
   a(i,1)=a(i+1,1);
   a(i+1,1) = temp;
   sorted = false;
end
end
end
t1=toc %Zaustavi štopericu

fprintf (1, 'Sortitanj niza koriscenjem Quick sort algoritma \n')
tic; %Pokreni štopericu
for i=1:n-1
   for j=i+1:n
      if b(i)>b(j)
         temp=b(i);
         b(i)=b(j);
         b(j)=temp;
      end;
   end;
end;
t2=toc %Zaustavi štopericu

fprintf (1, 'Sortitanje niza koriscenjem MATLAB sort algoritma \n')
tic; %Pokreni štopericu
c=sort(c);
t3=toc %Zaustavi štopericu
```

Да би се перформансе једног процесора могле квантитивно изразити, потребно је дефинисати јединице у којима би се те перформансе могле изражавати. Најједноставнији показатељ перформанси једног процесора је време извршавања програма. Време извршавања неког програма може се разложити на три параметра:

- Време трајања једног тактног интервала (*Clock Cycle Time*)
- Просечна дужина инструкције у циклусима(*Cycles Per Instruction - CPI*)
- Број инструкција у извршаваном програму (*Instruction Count*)

Прва два параметра, време трајања једног тактног интервала и просечна дужина инструкције у циклусима која утичу на брзину извршавања програма, везана су за технологију израде процесора као и за његову архитектуру па се помоћу њих може дефинисати један он најједноставнијих параметара перформанси једног процесора. Тај параметар представља просечну брзину извршавања инструкција у јединици времена и он се најчешће изражава у јединицама MIPS (Milion Instructions Per Second) и представља број милиона инструкција које један процесор изврши у једној секунди. Он се израчунава према следећој формули:

$$MIPS = \frac{1}{ClockCycle\left[\frac{seconds}{cycle}\right] * CPI\left[\frac{instructions}{cycle}\right]} = \frac{ClockFrequency\left[\frac{cycles}{second}\right]}{CyclesPerInstruction\left[\frac{instructions}{cycle}\right]} \quad (3)$$

Параметар просечног броја инструкција, није најпоузданији показатељ перформанси једног процесора јер резултати могу јако варирати од типа примењеног програма за тестирање, тј од типа коришћених инструкција. [2]

Пример 6. Посматрајмо процесор који поседује јединицу за рад у покретном зарезу. Уколико се за решавање неког софтверског проблема у покретном зарезу користе хардверске инструкције које користе ову јединицу имаћемо мали број дугачких инструкција које ће се извршити за одређено време. Уколико исти проблем покушамо решити софтверски имаћемо велики број малих инструкција које ће се извршавати знатно дуже у односу на хардверску реализацију проблема. Иако ће време извршавања бити дуже, софтверско решење ће имати већи вредност у односу на хардверско решење.

Пример 7. Компајлер је у стању да оптимизује код тако што може смањити број аритметичко-логичких операција за 50 %. Одредити вредност MIPS-а у случају неоптимизованог и оптимизованог кода.

Бенчмарк представља метод мерења перформанси у односу на неку референтну вредност дефинисану стандардом. Бенчмарк програми се могу поделити на:

- бенчмарк програми засновани на реалним апликацијама
- бенчмарк програми засновани на решавању математичких проблема
- синтетички бенчмарк програми.

Бенчмарк програми засновани на реалним апликацијама испитују време извршавања програма или неки други параметар преко којег је произвођач апликације омогућио мерење перформанси.

Математички бенчмарк програми засновани су мерењу брзине извршавања одређених математичких проблема као што су: налажење броја PI коришћењем Monte Carlo методе, налажење простих бројева помоћу Ерестотеновог сита, решавање система линеарних једначина и друго.

Пример 8. Ератостеново сито представља алгоритам за налажење простих бројева мањих од неког задатог броја *n*, који се може употребити како би се изразиле перформансе процесора и компајлера у којем је програм написан.

Алгоритам:

- Напишите све бројеве од 2 до n
- Почевши од првог броја на списку (двојка) прецртајте са списка све бројеве дељиве са два и упишите да је двојка прост број.
- Понављајте поступак са следећим непрецртаним бројем m. Дакле, прецртајте све бројеве дељиве са m, а њега самог обележите да је прост.

Оптимизација алгоритма

- Поступак прецртавања бројева почињемо од m^2 јер су сви бројеви мањи од m^2 већ избрисани.
- Чим нађемо прост број m такав да је $m^2>n$ немамо потребу за даљим брисањем. Сви преостали бројеви су прости! [10]

Пример у Матлабу:

```
clc
clear all;

n=100;
%kreira se niz brojeva 2,3,4...n sa n-1 clanova
A=linspace(2,n,n-1);
%pomocni niz: 1 ako je broj prost, 0 ako nije prost
B=ones(1,n-1);

 tic; %Pokreni štopericu
 for m=1:n-1
    if B(m)==1     %ako je broj prost
       for j=2:(floor(n/A(m)))
          B(A(m)*j-1)=0; %izbaci sve njegove umnoske
       end
    end
 end
```

```
t=toc %Zaustavi štopericu

C=zeros(1,sum(B)); %niz prostih brojeva
i=1;
for j=1:n-1
   if B(j)==1  %ako je broj prost
      C(i)=A(j);  %kopiraj ga na i-to mesto u nizu C
      i=i+1;
   end
end
```

Најкоришћенији параметар за класификовање перформанси процесора је број милиона операција у покретном зарезу, који се изражава у јединицама MFLOPS (Milion Flopatin Point Operations Per Second). Помоћу овог параметра најчешће се оцењују перформансе великих суперрачунара који се користе за комплексна израчунавања. Један од најједноставнијих начина за одређивање овог параметра је коришћење LINPACK бенчмарка, помоћу кога се мери време решавања система линеарних једначина са *n* непознатих. Решавањем система линеарних једначина од n непознатих облика Ax=B, при чему су матрице A и B димензија $n_x n$ и $n_x 1$, респективно, извршиће се $\frac{2}{3}n^3 + 2n^2$ операција у покретном зарезу.

Синтетички бенчмарк програми су специјалне апликације које своји извршавањем опонашају уобичајен рад на рачунару. Ове апликације на основу брзине извршавања генеришу одређене параметре који се могу користити за поређење перформанси рачунарских система.

Dhrystone представља синтетички програм за одређивање перформанси централног процесора приликом решавања проблема системског програмирања (целобројни рачун). Развијен је 1984. године од стране Reinhold Wiucker-a и постао је репрезентативан тест за перформансе процесора. Бенчмарк се састоји од мешавине математичких и операција за рад са стринговима. Резултат Dhrystone benchmarka се изражава у броју итерација које је процесор успео да изврши у току једне секунде. Резултати Dhrystone benchmarka се најчешће изражавају у јединицама DMIPS (Dhrystone MIPS) које се добијају када се број Dhrystone итерација у једној секунди подели са 1757 (број Дхрустоне итерација у секунди процесора VAX11/780, који поседује перформансе од 1 MIPSa). Тренутно актуелна верзија Dhrystone benčmarka је 2.1[11]

Предности Dhrystone алгоритма:

- Код је написан у C језику – може се извршавати на великом броју платформи
- Код је веома мали – лако се разуме и учи
- Перформансе се могу изразити у виду једног параметра DMIPS
- Добар за 8-битне и 16-битне микроконтролере.

Мане Dhrystone алгоритма:

- Третира само перформансе централног процесора
- Мали скуп операција, тестира искључиво целобројне операције
- Ослања се на стандардне C функције
- Benchmark не еволуира
- Поред процесора посматрају се перформансе и компајлера
- Оптимизацијом компајлера могу се добити драстично бољи резултати
- Benchmark није сертификован
- Не постоји стандардизована процедура за тестирање.

Начини да се превари Dhrystone benchmark:

- Слагати верзију benchmarka (написати резуктате из 1.1 или 2.0 уместо 2.1)
- Креирати специјанле бибилиотеке за рад са стринговима
- Користити оптимизацију компајлера.

Whetstone представља синтетички бенчмарк за одређивање перформанси централног процесора приликом комплексних израчунавања у покретном зарезу. Овај бенчмарк се састоји од мешавине математичких операција са мартицама, тригонометријских и експоненцијалних операција. Резултат Whetstone бенцхмарка се изражава у броју итерација које је процесор успео да изврши у току једне секунде. Резултати Wхетстоне бенцхмарка се најчешће изражавају у јединицама WMIPS (Whetstone MIPS) које представљају број Miliona Whetstone итерација у једној секунди.

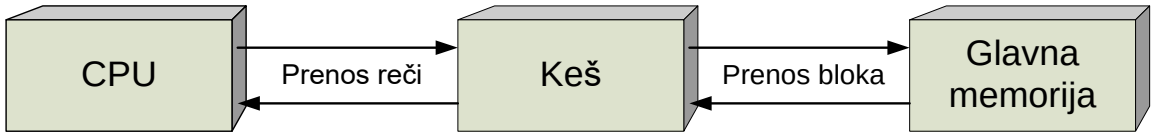
2. ПЕРФОРМАНСЕ МЕМОРИЈСКОГ СИСТЕМА РАЧУНАРА

Меморијски подсистем рачунара служи за складиштење бинарних информација и састоји се од низа меморијских компоненти, које се међусобно разликују по капацитету и брзини времена приступа. Главни делови меморијског подсистема су:

- регистри процесора
- кеш меморија
- оперативна меморија
- спољна меморија

2.1. Перформансе кеш меморије

Кеш меморија је транспарентно имплементирана између процесора и меморије и улога је да премости јаз који се јавља услед велике разлике у брзини рада процесора и оперативне меморије. Она представља меморију малог капацитета и велике брзине приступа, реализовану у SRAM (Static RAM) техници.



Слика 4. Принцип рада кеш меморије[1]

Кеш меморија садржи копију дела главне меморије, тј. један део блокова на које је подељена главна меморија. Када процесор жели да прочита одређену реч из оперативне меморије, прво се проверва да ли се та реч налази у неком од блокова у кеш меморији. Ако се њен блок налази у кеш меморији, та реч се испоручује процесору. Ако не, тада се одговарајући блок главне меморије који садржи тражену реч преноси у кеш меморију и процесору се испоручује захтевана реч.

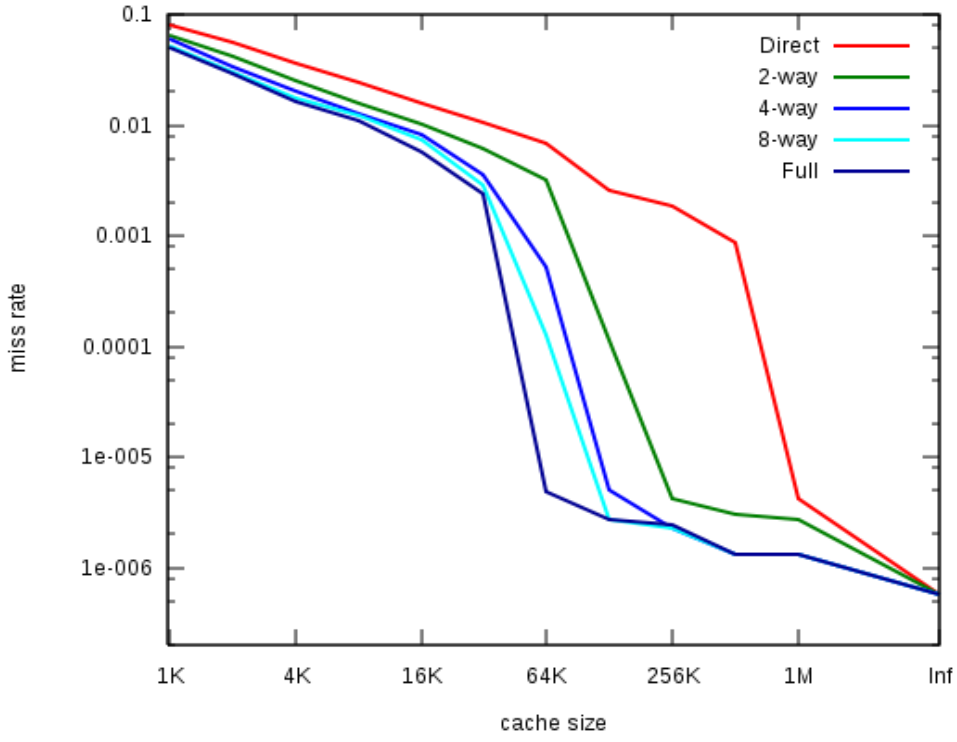
Код процесора старијих генерација, меморијски подсистем је имао углавном једну кеш меморију која је била лоцирана ван процесора и процесор је са њом комуницирао преко релативно споре спољашње магистрале. Како се временом резолуција израде процесора повећавала, стекле су се могућности да се кеш налази на истом чипу као и процесор, чиме се значајно повећава брзина комуникације са кеш меморијом. Такође временом се повећавао број нивоа кеш меморије у меморијском подсистему, тако да садашњи савремени процесори имају кеш меморију у три нивоа, L1, L2, L3 и оне су углавном све интегрисане на истом чипу као и процесор.

Принцип увођења кеш меморије у меморијски подсистем, омогућује да он поседује капацитет оперативне меморије, при чему је брзина приступа приближно једнака брзини приступа кеш меморији. Оваква организација оперативне меморије, назива се меморија у два нивоа. Основни услов да би овај концепт функционисао је принцип локалности референце. Тај принцип тврди да меморијске референце теже да се групишу. У дугом

временском периоду, групе у употреби се мењају, али у кратком времену, процесор ради првенствено са фиксираним групама меморијских референци.

- Изузев за инструкције гранања и позива, који сачињавају као мали део свих програмских инструкција, извршење програма је секвенцијално. Дакле у већини случајева следећа инструкција која треба да се донесе непосредно прати претходно донесену инструкцију.
- Појава дугачких непрекинутих секвенци позива процедура је редак случај, па програм остаје затворен у веома уском оквиру дубине позива процедура.
- Већина итеративних конструката састоји се од релативно малог броја инструкција које се понављају много пута. За време итерације прорачун је затворен у малом суседном делу програма.
- У многим програмима, велики део прорачуна обухвата обраду структура података, као што су матрице или секвенце записа. У многим случајевима узастопне референце на те структуре података биће блиско лоциране ставке података.

Промашај представља ситуацију када се тражени блок не налази у кеш меморији па је потребно његово учитавање из главне меморије. Уколико имамо мали број промашаја, процесор ће мање времена трошити чекајући спору оперативну меморију, тј. жељене референце ће добијати из брзе кеш меморије.



Слика 5. Промашаји у кешу као функција величине кеш меморије и начина пресликавања[12]

Промашаје у кеш меморији можемо класификовати у три типа промашаја:

- Принудни (compulsory) промашаји су промашаји који се јављају приликом првог прозивања референце. Они не зависе од капацитета кеш меморије као ни од типа коришћеног пресликавања у кеш меморији, али зависе од величине блокова у кеш меморији. Промашаји овог типа могу се избећи.

- Промашаји услед утицаја капацитета су губици на које не утичу тип коришћеног пресликавања као ни величина блока већ само величина кеш меморије.

- Конфликтни промашаји настају приликом замене блокова у кеш меморији и могу се поделити у две подгрупе:

- Промашаји услед мапирања су промашаји који су неизбежни за одређену технику пресликавања

- Промашаји услед замене блока настају приликом избора блока који ће бити замењен у кеш меморији.

2.1.1. Рад меморије у два нивоа

Меморијски систему два нивоа се састоји меморије горњег нивоа M1, мале брзе кеш меморије са временом приступа T1 и меморије доњег нивоа M2, оперативне меморије са временом приступа T2. Када се направи меморијска референца, прво се покушава приступ тој референци у меморији горњег нивоа M1. Ако се та референца пронађе, време приступа износи T1. Ако се референца не пронађе у горњем нивоу, онда се одговарајући блок копира из доњег нивоа M2 у горњи ниво M1 и затим се та референца испоручује процесору, а време приступа у том случају је T1+T2. Због ефекта локалности референце, једном кад се блок донесе у M1, требало би да дође до извесног броја приступа локацијама у том блоку, што за резултат има бржи укупни приступ. Ако са H обележимо вероватноћу да се тражена меморијска референца нађе у меморији горњег нивоа, тада је просечно време приступа једнако: [1]

$$T_S = H \cdot T_1 + (1-H) \cdot (T_1 + T_2) = T_1 + (1-H)T_2 \qquad (4)$$

Рад меморије у два нивоа приказаћемо у следећим примерима:

Пример 9: Одредити просечно време приступа меморији у два нивоа, ако је време приступа горњем нивоу 500ps, време приступа доњем нивоу 4,7ns а вероватноћа проналажења референце у кеш меморији износи 0.91.

$$T_S = H \cdot T_1 + (1-H) \cdot (T_1 + T_2) =$$
$$T_S = 0.91 \cdot 500 \cdot 10^{-12} + (1-0.91) \cdot \left(500 \cdot 10^{-12} + 4.7 \cdot 10^{-9}\right) = 923 \cdot 10^{-12} = 923ps$$

Ефикасност приступа меморијском систему у два нивоа представља однос времена приступа горњем меморијском нивоу T1 и просечног времена приступа Ts и оно се најчешће изражава у функцији вероватноће поготка.[1]

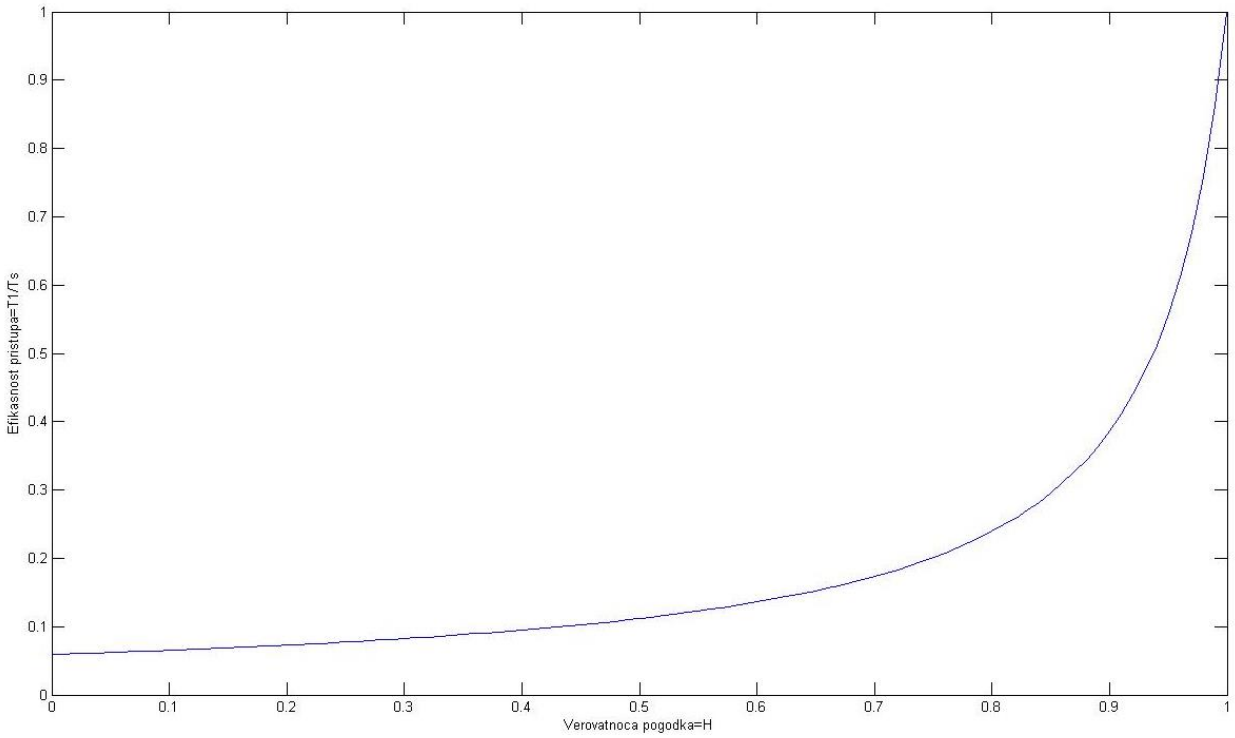
$$\frac{T_1}{T_S} = \frac{1}{1 + (1-H) \cdot \frac{T_2}{T_1}} \tag{5}$$

Пример 10: Нацртати графикон ефикасност приступа меморији у два нивоа у функцији вероватноће поготка H, ако време приступа меморији вишег нивоа износи 1,7ns а време приступа меморији нижег нивоа износи 27ns.

```
clc;
clear all;
clf;

n=100;
T1=1.7e-9;
T2=27e-9;

H=0:1/n:1;
Kes=zeros(1,n+1);
for i=1:n+1
Kes(i)=1/(1+(1-H(i))*T2/T1);
end
plot(H,Kes(1,:))
```



Слика 6. Ефикасност приступа у зависности од вероватноће поготка

2.2. Перформансе оперативне меморије

Оперативна меморија представља скуп интегрисаних кола која служе за памћење бинарних података. Њене две основне карактеристике су: равноправност приступа и привременост памћења. Под равноправношћу приступа подразумевамо да је време приступа било којој меморијској локацији исто. Привременост памћења је карактеристика полупроводничких меморија да по нестанку напона напајања губе све запамћене податке. По начину реализације меморијских ћелија меморије се деле на статичке меморије (static RAM) и динамичке меморије (dynamic RAM). Статичке меморије за свој рад користе флип-флопове који се реализују помоћу четири до шест повезаних транзистора. Динамичке меморије свој рад базирају на стању напуњености кондензатора који представља основну меморијску ћелију. Предност статичких меморија је у брзини рада а динамичких у капацитету и густини паковања. Мана динамичких меморија је неопходност освежавања садржаја меморије, јер се кондензаторске ћелије временом празне услед паразитних капацитивности.

У току историје развоја рачунара, на тржишту се користило више типова меморија:

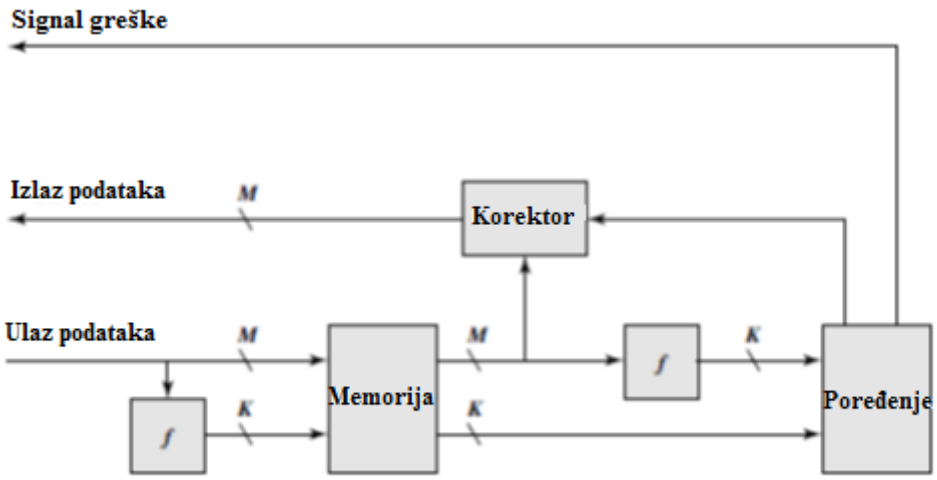
- Fast-page mode memory (FPM)
- Extended data output (EDO)
- Synchronous DRAM memory
- Direct RAM bus (RDRAM)

Параметри који се користе за дефинисање оперативне меморије су:

1. Капацитет оперативне меморије представља укупну количину информација која се може ускладиштити у меморији и најчешће се изражава у бајтима или битима, мада се може изражавати и у речима где једна реч представља капацитет једне адресибинле локације
2. Време приступа оперативној меморији дели се на:
   а. време читања (read time) $\mathbf{t_r}$ представља најкраћи временски интервал између два узастопна читања једне исте меморијске локације.
   б. време уписа (write time) $\mathbf{t_w}$ представља најкраћи временски интервал између два узастопна у исту меморијску локацију.
3. Меморијски цилкус (cycle time) $\mathbf{T_c}$ представља просечно време два узастопна читања и писања у исту меморијску локацију и израчунава се по следећој формули.
4. Дужина податка W који се може прочитати/уписати у меморију у једном временском тренутку и изражава се у бајтовима или битима
5. Брзина преноса меморије (memory transfer rate) дели се на:
   а. брзину уписа података у меморију (memory write speed)
   б. брзину читања података из меморије (memory read speed)

2.2.1 Исправљање грешака у оперативној меморији

Полупроводничка меморија је подложна грешкама. Оне могу да се окарактеришу као хардверске и привремене. Хардверска грешка је трајан физички квар, тако да меморијска ћелија или ћелије које су њом обухваћене не могу поуздано да складиште податке, него се закоче на 0 или 1, или погрешно прескачу између 0 и 1. Хардверске грешке могу да буду проузроковане оштрим повредама из околине, грешкама у производњи и истрошеношћу. Привремена грешка је случајан, недеструктиван догађај, који мења садржај једне или више меморијских ћелија, без оштећења меморије. Привремене грешке могу да буду проузроковане проблемима у напајању електричном енергијом, или алфа честицама. Те честице резултују из радиоактивног распадања и, нажалост, веома су присутне, зато што се радиоактивна језгра у малим количинама налазе у скоро свим материјалима. Услед тренда смањивања меморијских ћелија оне све више постају подложне чешћој појави привремених грешака па већина савремених меморијских система укључује логику, како за откривање, тако и за исправљање грешака.

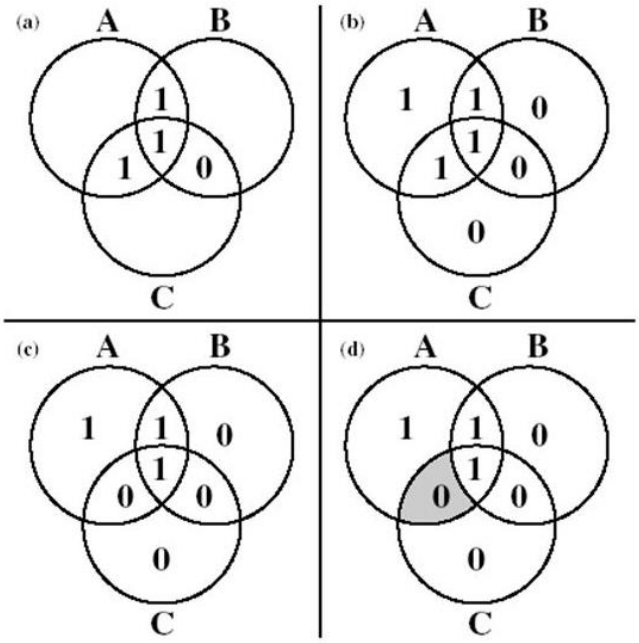


Слика 7. Провера и исправљање грешака у полупроводничкој меморији [1]

На слици је илустрован процес детекције и отклањања грешака у оперативној меморији. Приликом смештања података у меморију на основу њих се генерише код коришћењем функције кодирања *f* који се такође складишти у меморији. Дакле, ако треба да се ускладишти М – битна реч података, а код је дужине К битова, онда је стварна дужина ускладиштене речи М + К битова.

Приликом читања ускладиштених података, на основу њих се генерише код коришћењем исте функције кодирања *f*, који се пореди са битовима ускладиштеног кода. Поређење може да доведе до једног од следећа три резултата: 1. Није откривена никаква грешка. Донесени подаци се шаљу напоље. 2. Откривена је грешка и могуће је да се она исправи. Битови података и битови за исправљање грешке уносе се у коректор, који производи исправљен скуп од М битова да се пошаље напоље. 3. Откривена је грешка, али није могуће да се она исправи.

Један од најједноставнијих кодова за детектовање грешки је код парноси али овај код може само да детектује да у подацима постоји грешка али не може да утврди локацију грешке како би је отклонио. Најједноставнији од кодова за исправљање грешака је Хамингов код. На слици 8 су употребљени су Венови дијаграми да би се илустровала употреба тог кода на 4 – битним речима (М = 4). Када имамо три круга А, В и С који се међусобно секу, постоји седам одељака. Доделићемо 4 бита података унутрашњим одељцима на пресеку кругова (слика 8а). Спољашњи одељци се попуњавају битовима парности. Бит парности се поставља на нулу је тако да је укупни број цифара 1 у његовом кругу паран и обратно (слика 8б). Стога, бит парности круга А поставља се поставља на јединицу јер у том кругу имамо непаран број јединица података (три) а битови парности у круговима В и С се постављају на нулу јер у тим круговима имамо паран број јединица података (по два).



Слика 8. Провера и исправљање грешака у полупроводничкој меморији[1]

Тако генерисан Хамингов код се складишти са битовима податка. Приликом провере над прочитаним битовима података се поново генеришу битови Хаминговог кода који се пореде са ускладиштеним битовима. Као резултат поређења могућа су четири сценарија:

1. сви битови прочитаног и генерисаног кода се поклапају што значи да у подацима не постоји грешка
2. само један бит прочитаног и генерисаног кода се не поклапају што значи да је грешка на биту Хаминговог кода и да на подацима не постоји грешка
3. два бита прочитаног и генерисаног кода се не поклапају што значи да је грешка настала биту податка који се налази на пресеку кругова на којима постоји непоклапање битова Хаминговог кода. Тај бит података је потребно инвертовати како би се отклонила грешка
4. сва три бита прочитаног и генерисаног кода се не поклапају што значи да је грешка настала биту податка који се налази на пресеку сва три круг. Тај бит података је потребно инвертовати како би се отклонила грешка.

Када се због грешке један од битова података промени (слика 8c), то се лако открива. Проверавањем бита парности, откривају се несагласности у круговима A и C, али не и у кругу B. Само један од седам одељака је у A и C, али не и у B. Грешка зато може да се исправи променом тог бита.

Описани код је познат као код за исправљање једне грешке (*single-error-correcting*, SEC). Полупроводничка меморија се чешће опрема кодом за исправљање једне и откривање две грешке (*single-error-correcting, double-error-detecting*, SEC-DED). Као што је показано у табели 2, такви кодови у поређењу са SEC кодовима захтевају један додатни бит.

Табела 2. Повећање дужине речи са исправљањем грешака[1]

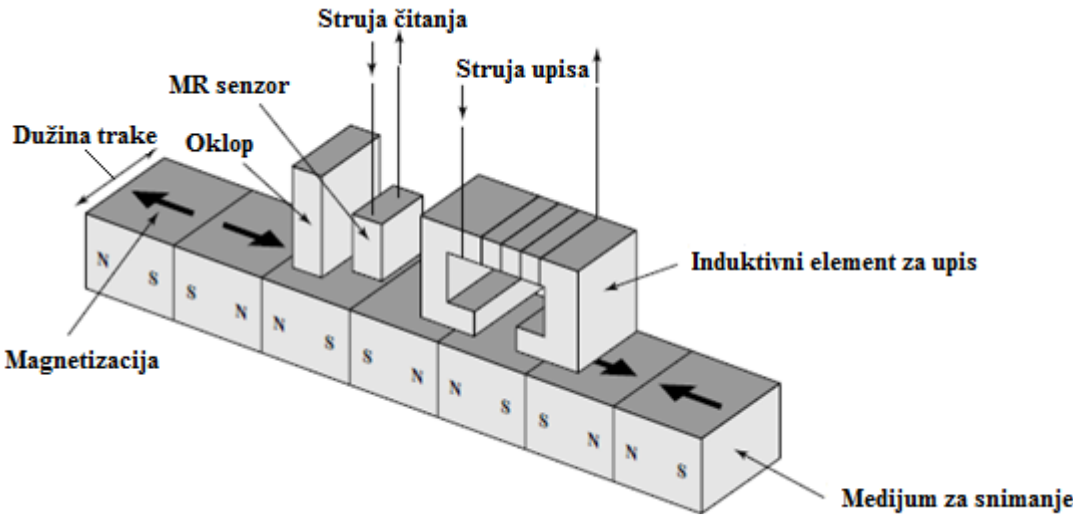
Исправљање једне грешке			**Исправљање једне грешке / откривање две грешке**	
Битови података	**Битови за проверу**	**% повећања**	**Битови за проверу**	**% повећања**
8	4	50,00	5	62,5
16	5	31,25	6	37,5
32	6	18,75	7	21,875
64	7	10,94	8	12,5
128	8	6,25	9	7,03
256	9	3,52	10	3,91

Код за исправљање грешака повећава поузданост меморије, по цени додатне сложености. Са организацијом од једног бита по чипу, код SEC – DED се обично сматра одговарајућим. На пример, имплементације IBM 30xx користиле су 8 – битни SEC – DED код за свака 64 бита података у главној меморији. Дакле, величина главне меморије је стварно око 12% већа него што то изгледа кориснику. Рачунари VAX су користили 7 – битни SEC – DED код за свака 32 бита меморије, што је додатних 22%. Велики број савремених DRAM меморија користи 9 битова провере за сваких 128 битова података, што је додатних 7%.[1]

2.3 Перформансе спољне меморије

Спољња меморија представља меморију намењену за трајно складиштење података. У спољње меморије спадају магнетни диск, флексибилни магнетни дискови и траке, оптичке меморије, трајне полупроводничке меморије FLASH и друго.

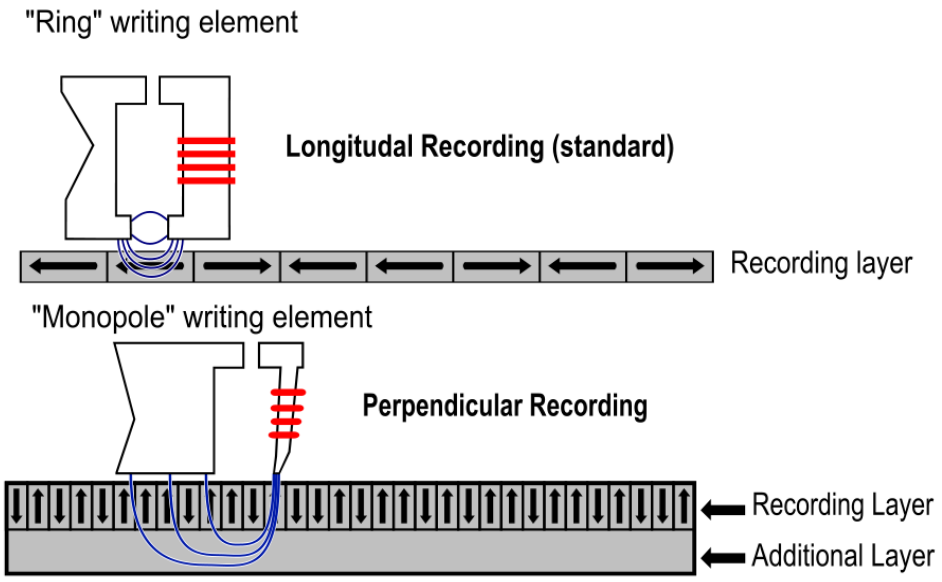
Магнетни диск се састоји од скупа кружних плоча израђених од немагнетног материјала (алуминијум, стакло), који је пресвучен танким слојем магнетног материјалом који се може намагнетисати и тако ускладиштити информације. Подаци се уписују и читају са магнетног диска преко проводног намотаја који се зове уписно-читајућа глава. Како би се побољшале перформансе у савременим магнетним дисковима постоје две посебне главе за читање и за уписивање. За време операције читања или уписивања, глава је стационарна у односу на површину диска која ротира испод ње. Механизам за уписивање заснован је на закону електромагнетне индукције; електрична струја ствара магнетно поље и променљиво магнетно поље индукује електромоторну силу у проводном намотају. Приликом уписа на магнетни диск, електрични импулси који одговарају информацијама које желимо уписати се шаљу у главу за уписивање. Уписна глава направљена је од материјала који се лако намагнетише и у облику је правоугаоног прстена са процепом дуж једне стране и неколико намотаја проводне жице дуж супротне стране. Уписна глава ствара магнетно поје које магнетише материјал на површини диска који се налази непосредно испод уписне главе. У зависности смера струје уписа, дефинише се смер магнетизације, односно да ли ће на површини диска бити уписана логичка нула или јединица.



Слика 9. Усписна и читајућа глава[1]

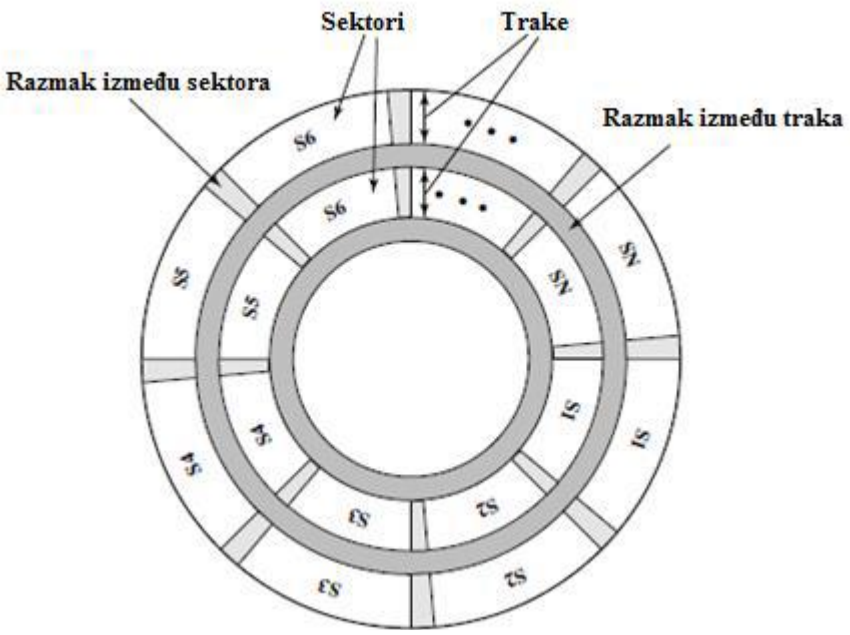
Приликом читања података са диска, намагнетисани материјал услед обртања диска ствара промењљиво магнетно поље у колу читајуће главе, што за последицу има индуковање електромоторне силе у намотајима читајуће главе. Смер индуковане електромоторне силе зависиће од смера магнетизације материјала на површини диска, односно зависиће од тога да ли је на диску уписана логичка нула или јединица. Читајућа глава се код савремених дискова чешће израђује у облику магнетно-резистивног или Hall сензора, како би се поједноставила конструкција и побољшале перформансе.

Капацитет магнетног диска зависи од густине записивања информација, па се временом величина магнетних елемената знатно смањивала. Пошто је код планарне организације магнетних елемената, њихова величина ограничена величином уписно-читајуће главе, произвођачи магнетних дискова су прешли на вертикалну организацију магнетних елемената чиме је се знатно повећала густина записивања. Магнетна глава је променила облик и састоји се од једног зашиљеног дела који магнетише врло узак део магнетног материјала. Испод магнетног слоја налази се слој меког магнетног материјала кроз који се затвара магнетно поље. Магнетно поље се затим усмерава кроз исто магнетисане елементе (при чему се не мења њихов садржај) и затвара се кроз релативно велику стопу уписно-читајуће главе.



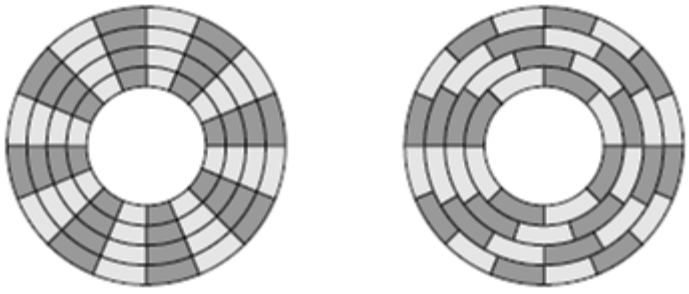
Слика 10. Планарно и вертикално уписивање података на магнетни диск[13]

Пошто су плоче магнетног диска кружног облика настаје проблем организације података на површини диска. Пошто плоча диска ротира испод уписно-читајуће главе, испод ње се формира прстен ширине уписно-читајуће главе који називамо стаза. Површина диска од центра ка периферији диска је подељена на низ концентричних стаза. Уписно читајућа глава се може померати у дискретним корацима при чему се позиционира изнад одговарајуће стазе. Како би се спречила интерференција између суседних стаза приликом операција уписа и читања, стазе су међусобно подељене размацима. Стазе су даље подељене у фиксан број сектора који представљају најмању јединицу података која се може уписати и прочитати са магнетног диска. У већини савремених система, користе се сектори фиксне дужине, где је 512 бајтова скоро универзална величина сектора.



Слика 11. Организација података на магнетном диску[1]

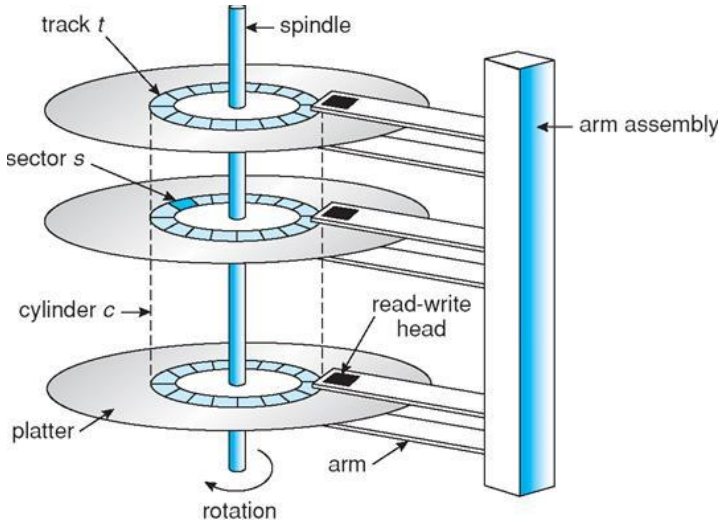
Оваква приступ организацији података на диску омогућава једнако време читања било ког сектора на диску јер се диск обрће константном угаоном брзином *(Constant Angular Velocity*, CAV). Недостатак ове технике је што је на спољашњим стазама знатно мања густина уписивања у односу на стазе у унутрашњости диска. Како би се овај недостатак могао отклонити користи се променљива брзина обртања диска у зависности од позиције стазе на површини диска. Приликом приступа стазама које се налазе ближе центру диска брзина обртања плоче ће бити већа у односу на брзину обртања ако се приступа стазама које се налазе на периферији диска. Овом технологијом се повећава капацитет магнетног диска али се знатно компликује организација података, као и управљање радом диска.



Слика 12. Организација података у зависности на угаону брзину диска[1]

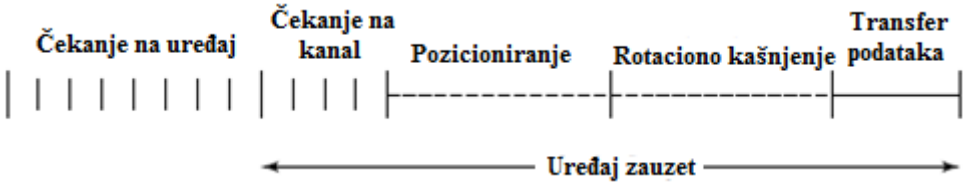
Како би се повећао капацитет магнетног диска, савремени магнетни дискови користе већи број магнетних плоча које су обострано намагнетисане. Плоче се механички

спрежу на исту осовину, тако да се заједно обрћу. Пошто свака површина плоче магнетног диска захтева једну уписно-читајућу главу, неопходно је да главе буду спрегнуте на заједнички позициони механизам, како би се поједноставила конструкција магнетног диска. Пошто се главе заједно крећу, оне формирају цилиндар који представља скуп стаза на које припадају различитим површинама плоча а налазе се на истом растојању од осе вратила.



Слика 13. Организација података у диску са већим бројем магнетних плоча[1]

Осим капацитета, веома важан параметар перформанси магнетних дискова је и време приступа подацима на диску. Ово време се може поделити на време позиционирања, време ротационог кашњења и време трансфера података.



Слика 14. Временски циклус код приступа подацима на магнетном диску[1]

Да би се приступило подацима на магнетном диску, уписно читајућа глава најпре мора да се позиционира на жељену стазу. Време које је потребно да би се глава поставила на стазу зове се **време позиционирања** (енгл. *seek time*). Ово време није константно и зависи од релативног положаја главе у односу на стазу којој се жели приступити. У таквом случају можемо разликовати следеће вредности времена позиционирања:

- минимално време позиционирања (*track-to-track seek time*) представља време потребно да уписно читајућа глава пређе на суседну стазу
- максимално време позиционирања (*full stroke seek time*) представља време потребно да уписно читајућа глава преже пут о прве до последње стазе на површини диска

- просечно време позиционирања (*average seek time*) представља просечно време позиционирања које се одређује на основу праћења статистике приступа подацима на диску.

Првобитно су главе покретане навојним вретеном управним на осу диска које је покретао корачни мотор. Овако решење је поседовало споро време одзива од 80 до 120ms, па је убрзо прешло на ротациони актуаторски механизам који поседује знатно краће време одзива од 20ms.

Након позиционирања уписно-читајуће главе изнад жељене стазе, неопходно је сачекати да се плоча диска заротира док почетак жељеног сектора на стази не дође испод уписно-читајуће главе. Време које је потребно да би почетак сектора стигао до главе зове се **ротационо кашњење** (енгл. *rotational latency*). Пошто ово време зависи од релативног положаја главе у односу на сектор којем се жели приступити, најчешће се узима просечно време ротационог кашњења које је једнако времену потребном да плоча диска направи половину пуног обртаја. Ово време је зависно од брзине обртања диска па су просечна времена ротационог кашњења за различите брзине обртања дискова дата следећом табелом:

Табела 3. Зависност просечног ротационог кашњења од брзине обртања диска

Брзина обртања диска [min^{-1}]	**Просечно ротационо кашњење [ms]**
4,200	7.14
5,400	5.56
7,200	4.17
10,000	3.00
15,000	2.00

2.3.1 Заштита података на спољној меморији

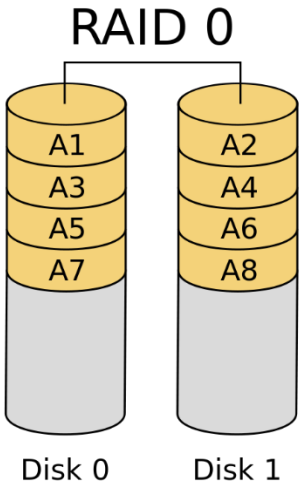
Брзина побољшања перформанси магнетних дискова током развоја рачунарских система у многоме је била мања од брзине развоја процесора и осталих компоненти, првенствено због физичких и механичких ограничења. Та неусаглашеност учинила је да се посебна пажња посвети систему складиштења података на магнетним дисковима како би се побољшала укупне перформансе рачунарског система. Конструктори су увидели да се перформансе магнетних дискова могу унапређивати само до извесне границе и да се додатни добици перформанси могу остварити низова дискова који раде независно и паралелно. Помоћу овог приступа засебни U/I захтеви могу да се опслужују паралелно, све док тражени подаци стоје на засебним дисковима. Поред тога, један U/I захтев може да се извршава паралелно ако је блок података коме треба да се приступи распоређен на више дискова.

Стандардизована шема за рад са вишеструким паралелним дисковима познатија као RAID (Redundant Array of independent disks) усвојена је као стандард од већине произвођача. Постоји седам RAID нивоа (од RAID 0 до RAID 7) који се међусобно разликују по организацији ускладиштења података на дисковима али имају три заједничке карактеристике:

1. RAID је скуп физичких дискова које оперативни систем види као један логички диск
2. Подаци су распоређени на физичким уређајима једног низа
3. Капацитет редундантног диска користи се за складиштење информације парности, што гарантује обновљивост података у случају отказа неког од дискова у низу.[1]

RAID **Ниво 0**

RAID Ниво 0 није прави члан RAID породице јер не укључује редундантност како би се повећала поузданост. Његове првенствене перформансе су у повећању капацитета и брзине приступа. Подаци који се смештају на RAID низ дискова се деле се на стазе које се распоређују по расположивим дисковима. У низу од *n* дискова, првих *n* логичких стаза се физички складиште као прва стаза на сваком од *n* дискова, других н стаза се расподељују као друге стазе на свим дисковима и тако даље. Предност оваквог распореда је томе да се за један U/I захтев који се састоји од више логички суседних стаза, истовремено опслужује н стаза чиме се у значајној мери смањује време U/I преноса.[1]



Слика 15. Организација података у RAID 0 систему[14]

Капацитет система једнак је суми капацитета дискова:

$$C_{RAID_0} = \sum_{i=1}^{n} C_{MD_i}$$

Вероватноћа отказа q_{RAID0} целог система једнака је суми вероватноћа отказа q_i појединих магнетних дискова:

$$p_{otkaza_{RAID0}} = \sum_{i=1}^{n} p_{otkaza_{MD_i}}$$

Вероватноћа исправног рада p_{RAID0} се добија одузимањем вероватноће отказа q_{RAID0} система од 1, а према следећој формули:

$$1 - p_{otkaza_{RAID0}} = \prod_{i=1}^{n} (1 - p_{otkaza_{MD_i}})$$

RAID Ниво 1

Код овог нивоа првенствени акценат је дат на редундантност ускладиштених података која се постиже једноставним копирањем стаза на два посебна физичка диска, тако да сваки диск у низу има свој диск - одраз у огледалу, који садржи исте податке. RAID 1 такође може да се имплементира и без дељења података на траке мада је то мање уобичајено. Постоји неколико позитивних аспеката организације RAID 1:

1. Захтев за читање може да се опслужи помоћу било ког од два диска који садрже тражене податке, који има минимално време позиционирања плус ротационо кашњење.
2. Захтев за уписивање тражи да обе одговарајуће траке буду ажуриране, ах то може да се уради паралелно. Према томе, перформансу уписивања диктира спорије од два уписивања (односно, оно које обухвата веће време позиционирања, плус ротационо кашњење). Међутим, нема „казне уписивања" код RAID 1. RAID нивои 2 до 6 обухватају употребу битова парности. Зато, када се ажурира једна трака, софтвер за управљање низом мора прво да прорачуна и ажурира битове парности, као и ажурирање стварне траке која је у питању.
3. Опоравак од отказа је једноставан. Када уређај откаже, подацима одмах може да се приступи на другом уређају.

Главни недостатак RAID 1 је цена; он захтева двоструки простор на дисковима за логички диск који подржава. Због тога се конфигурација RAID 1 вероватно ограничава на уређаје који складиште системски софтвер и податке, као и друге критичне датотеке. У тим случајевима, RAID 1 обезбеђује резервну копију у реалном времену за све податке, па у случају отказа диска, сви критични подаци су и даље одмах расположиви.

У окружењу оријентисаном на трансакције, RAID 1 може да постигне велике брзине U/I захтева ако се њихов највећи део односи на читање. У тој ситуацији, перформанса RAID 1 може да се приближи двострукој вредности оне коју има RAID 0. Међутим, ако је значајан део U/I захтева за уписивање, онда добици у односу на RAID 0 могу бити безначајни. RAID 1 може такође да обезбеди побољшану перформансу у односу на RAID 0 за апликације са интензивним преносом података, са великим процентом

читања. До побољшања долази ако апликација може да подели сваки захтев за читање, тако да оба диска члана могу да учествују у њиховом опслуживању.[1]

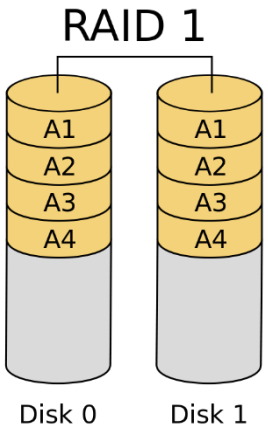
Код RAID 1 система, подаци се идентично записују на све дискове у низу, при чему се омогућава рад система са бар једним оперативним диском.

Капацитет RAID 1 система је једнак капацитету најмањег диска:

$$C_{RAID_1} = \min(C_{MD_i})$$

Вероватноћа отказа q_{RAID0} RAID 1 система је једнака производу вероватноћа отказа свих дискова q_i:

$$q_{RAID1} = \prod_{1=1}^{n} q_i$$



Слика 16. Организација података у RAID 1 систему[14]

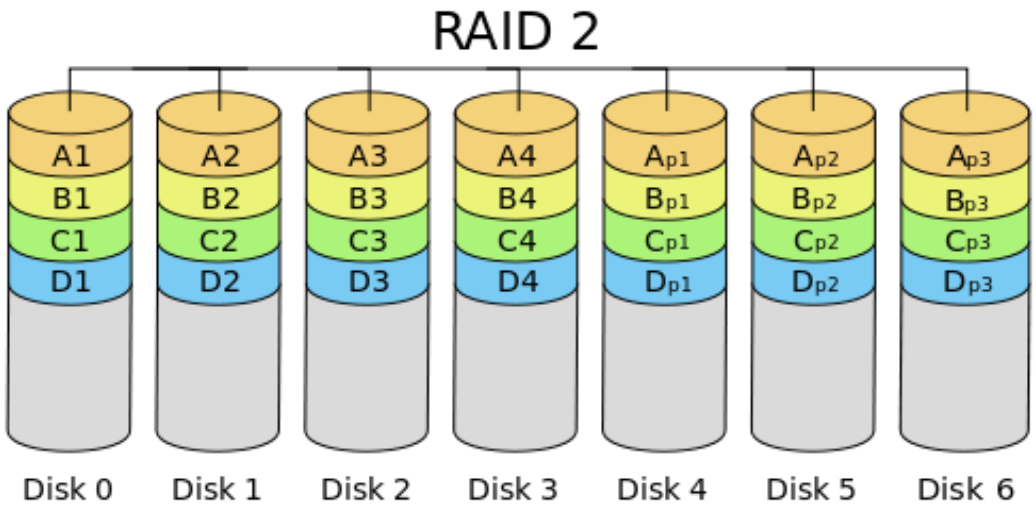
RAID Ниво 2

RAID ниво 2 користи паралелну технику приступа. У низу са паралелним приступом, сви дискови чланови учествују у извршењу сваког U/I захтева. Вретена појединачних дискова су типично синхронизована, тако да је свака глава диска у истом положају на сваком диску у било ком тренутку времена. Код овог нивоа подаци се деле у врло мале траке, дужине неколико бајтова које се паралелно уписују на низ дискова. Поред диска за податке постоји и диск на коме се чува Hammingov код који је у стању да исправи једноструке и детектује двоструке грешке битовима.

Иако RAID 2 захтева мање дискова од RAID 1, он је и даље веома скуп. Број редундантних дискова је пропорционалан логаритму броја дискова са подацима. Приликом једног читања, истовремено се приступа свим дисковима. Захтевани подаци и придружени код за исправљање грешака се испоручују контролеру низа. Ако постоји

једнострука грешка бита, контролер може да препозна грешку и одмах је исправи, тако да се време читања не продужава. Приликом једног уписивања, мора да се приступи свим дисковима за податке и за парност да би се обавила операција уписивања.

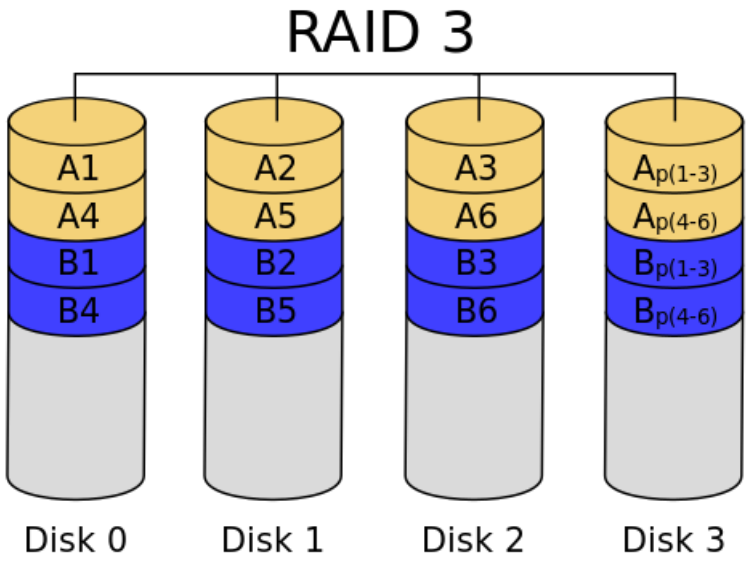
RAID 2 би могао да буде ефикасан избор само у окружењу у коме се појављује много грешака диска. Имајући у виду поузданост појединачних дискова и њихових уређаја, RAID 2 се ретко користи.[1]



Слика 17. Организација података у RAID 2 систему[14]

RAID ниво 3

RAID 3 се организује на сличан начин као RAID 2. Разлика је у томе што RAID 3 захтева само један редундантни диск, без обзира на то колико је велики низ дискова. Уместо кода за исправљање грешака, прорачунава се једноставан бит парности за скуп појединачних битова на истој позицији на свим дисковима са подацима.[1]



Слика 18. Организација података у RAID 3 систему[14]

RAID ниво 4

RAID нивои 4 до 6 користе технику независног приступа. У низу са независним приступом, сваки диск члан ради независно, тако да засебни U/I захтеви могу да се задовољавају паралелно. Због тога су низови са независним приступом погоднији за апликације које траже велике брзине U/I захтева, а релативно су мање погодни за апликације које траже велике брзине преноса података.

Као и у другим RAID шемама, користи се дељење података на траке. У случају RAID 4 до 6, траке су релативно велике. Код RAID 4, рачуна се бит по бит трака парности преко одговарајућих трака на сваком диску за податке, а битови парности се складиште у одговарајућој траци на диску за парност.

RAID 4 обухвата казну уписивања када се изводи U/I захтев за уписивањем малог обима. Сваки пут када се појави уписивање, софтвер за управљање низом мора да ажурира не само корисничке податке, него и одговарајуће битове парности. Замислите низ од пет уређаја у коме X0 до X3 садрже податке, а X4 је диск за парност. Претпоставите да се извршава уписивање које обухвата само траку на диску X1. У почетку, за сваки бит и, имамо следећи однос:

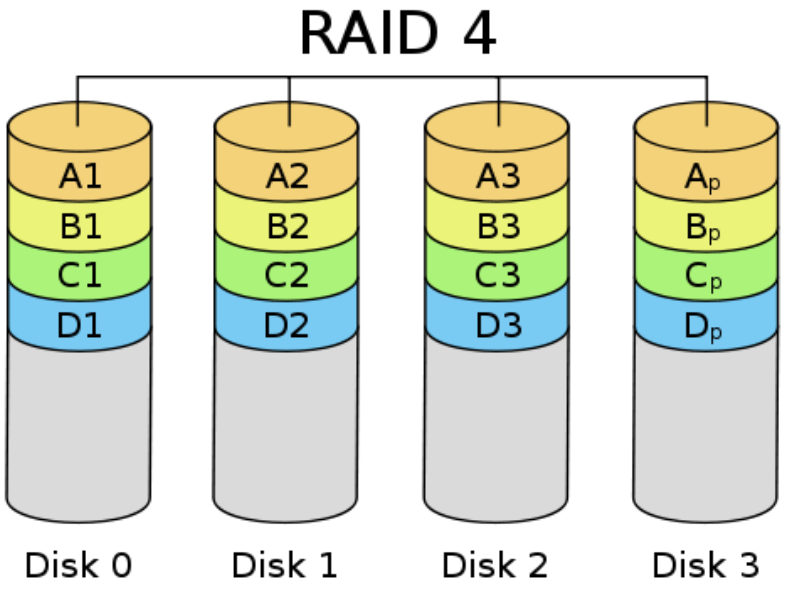
$$X4(i) = X3(i) \oplus X2(i) \oplus X1(i) \oplus X0(i)$$

После ажурирања, са потенцијално промењеним битовима означеним симболима „прим“ (’):

$$\begin{aligned} X4'(i) &= X3(i) \oplus X2(i) \oplus X1'(i) \oplus X0(i) \\ &= X3(i) \oplus X2(i) \oplus X1'(i) \oplus X0(i) \oplus X1(i) \oplus X1(i) \\ &= X3(i) \oplus X2(i) \oplus X1(i) \oplus X0(i) \oplus X1(i) \oplus X1'(i) \\ &= X4(i) \oplus X1(i) \oplus X1'(i) \end{aligned}$$

Претходни скуп једначина изводи се као што следи. Први ред показује да ће промена у X1 такође утицати и на диск парности X4. У другом реду, додајемо чланове $[\oplus X1(i) \oplus X1(i)]$. С обзиром на то даје XOR (ексклузивно-ILI) сваке величине са самом собом једнако 0, то не утиче на једначину. Међутим, то је погодност употребљена да би се направио трећи ред, праменом редоследа. Најзад, прва четири члана су замењена са X4(i).

Да би израчунао нову парност, софтвер за управљање низом мора да прочита стару корисничку траку и стару траку парности. Онда он може да ажурира те две траке са новим подацима и новоизрачунатом парношћу. Тако свако уписивање у траку обухвата два читања и два уписивања. У случају U/I већег обима, уписивања које обухвата траке на свим уређајима дискова, парност се лако израчунава коришћењем само нових битова података. На тај начин диск за парност може да се ажурира паралелно са уређајима за податке и нема додатних читања или уписивања. У сваком случају, свака операција уписивања мора да обухвати диск за парност, који зато постаје уско грло.[1]



Слика 19. Организација података у RAID 4 систему[14]

RAID 4 користи технику независног приступа где се подаци групишу у блокове величине једног или више сектора и уписују се на дискове у низу. Исправљање грешака се постиже коришћењем провере парности блокова и тај блок се складишти на засебном диску.

Капацитет RAID 4 система је једнак суми капацитета дискова који се користе за податке. Систем може реконструисати податке уколико откаже један од дискова.

Вероватноћа отказа q_{RAID4} код RAID 4 система се рачуна на следећи начин

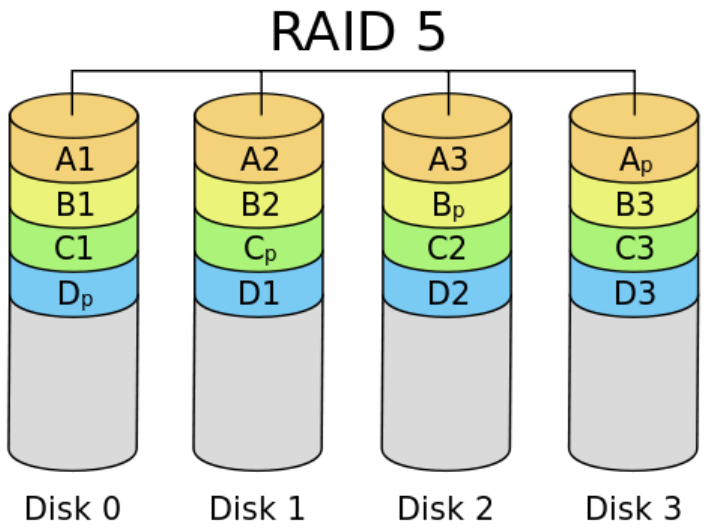
$$q_{RAID4} = q_1 \cdot q_2 \cdot (1-q_3) + q_1 \cdot (1-q_2) \cdot q_3 + (1-q_1) \cdot q_2 \cdot q_3 + q_1 \cdot q_2 \cdot q_3$$

Вероватноћа исправног рада p_{RAID4} се добија према следећој формули:

$$p_{RAID4} = p_1 \cdot p_2 \cdot (1 - p_3) + p_1 \cdot (1 - p_2) \cdot p_3 + (1 - p_1) \cdot p_2 \cdot p_3 + p_1 \cdot p_2 \cdot p_3$$

RAID ниво 5

RAID 5 се организује на сличан начин као RAID 4. Разлика је у томе што RAID 5 расподељује траке за парност по свим дисковима. Типично, подела је по шеми са кружним додељивањем. За низ од н дискова, трака за парност је на различитом диску за првих н трака, а онда се узорак понавља. Расподелом трака за парност по свим уређајима, избегава се потенцијално U/I уско грло које се налази у RAID 4.[1]

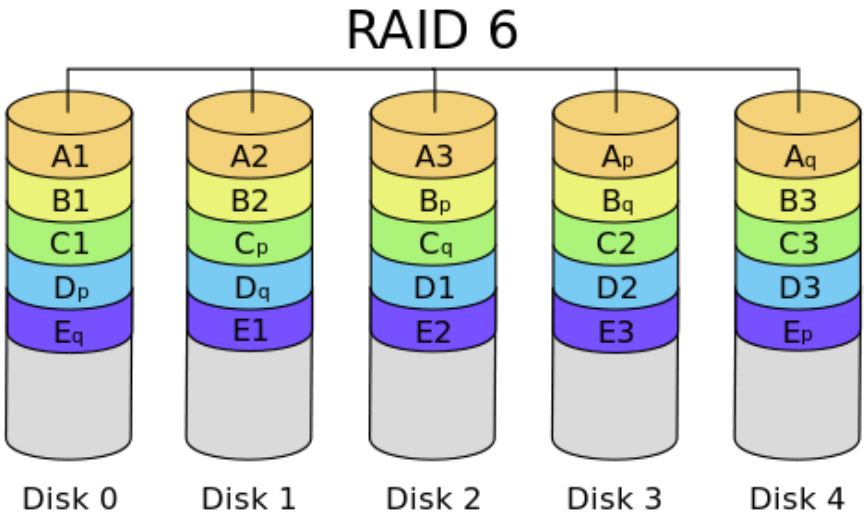


Слика 20. Организација података у RAID 5 систему[14]

RAID ниво 6.

У шеми RAID 6, врше се два различита прорачуна парности и складиште се у засебним блоковима на различитим дисковима. Зато се низ RAID 6, чији кориснички подаци захтевају N дискова, састоји од N + 2 диска. P и Q су два различита алгоритма за проверу података. Један од њих је прорачун екскључивног-ILI који се користи у RAID4 и 5. Али други је независан алгоритам за проверу података. То омогућава да се обнове подаци чак и ако откажу два диска који садрже корисничке податке.

Предност RAID 6 је у томе што он обезбеђује веома високу расположивост података. Требало би да откажу три диска унутар интервала MTTR (средњег времена за оправку) да би се изгубили подаци. С друге стране, RAID 6 уноси значајну казну уписивања зато што свако уписивање утиче на два блока парности.[1]



Слика 21. Организација података у RAID 6 систему [14]

Пример 13. RAID систем се састоји од три независна диска са следећим карактеристикама:

Диск	**Капацитет (GB)**	**Вероватноћа отказа**
1	40	0,03
2	40	0,03
3	40	0,03

Одредити капацитет и вероватноћу отказа у случају:
а) RAID 0,
б) RAID 1,
ц) RAID 4, где се последњи диск користи за складиштење бита парности.

Решење:

a) $C_{RAID0} = \sum_{i=1}^{3} C_i = 40 + 40 + 40 = 120GB$

$$q_{RAID0} = \sum_{i=1}^{3} q_i = 0{,}03 + 0{,}03 + 0{,}03 = 0{,}09 = 9\%$$

b) $C_{RAID1} = \min(C_i) = 40GB$

$$q_{RAID1} = \prod_{1=1}^{n} q_i = 0{,}03 \cdot 0{,}03 \cdot 0{,}03 = 0{,}000027$$

c) $C_{RAID0} = \sum_{i=1}^{2} C_i = 40 + 40 = 80GB$

$$q_{RAID0} = q_1 \cdot q_2 \cdot (1 - q_3) + q_1 \cdot (1 - q_2) \cdot q_3 + (1 - q_1) \cdot q_2 \cdot q_3 + q_1 \cdot q_2 \cdot q_3 = 0{,}02646$$

Пример 14. RAID систем се састоји од три независна диска са следећим карактеристикама:

Диск	Капацитет (GB)	Поузданост (%)
1	100	99
2	150	98
3	200	98,5

Одредити капацитет, време приступа и вероватноћу отказа у случају:

а) RAID 0,

б) RAID 1,

ц) RAID 4, где се последњи диск користи за складиштење бита парности.

Решење:

a) $C_{RAID0} = \sum_{i=1}^{3} C_i = 100 + 150 + 200 = 450GB$

$$q_{RAID0} = 1 - p_{RAID0} = 1 - \prod_{i=1}^{n} p_i = 1 - p_1 \cdot p_2 \cdot p_3 = 1 - 0{,}99 \cdot 0{,}98 \cdot 0{,}985$$
$$= 1 - 0{,}955647 = 0{,}044353$$

b) $C_{RAID1} = \min(C_i) = 100GB$

$$q_{RAID1} = \prod_{i=1}^{n} q_i = (1 - p_1) \cdot (1 - p_2) \cdot (1 - p_3) = 0{,}02 \cdot 0{,}01 \cdot 0{,}015 = 0{,}000003$$

c) $C_{RAID4} = \sum_{i=1}^{2} C_i = 100 + 150 = 250GB$

$$q_{RAID4} = 1 - p_{RAID4} = 1 - p_1 \cdot p_2 \cdot (1 - p_3) + p_1 \cdot (1 - p_2) \cdot p_3 + (1 - p_1) \cdot p_2 \cdot p_3 + p_1 \cdot p_2 \cdot p_3 = 6{,}44 \cdot 10^{-4}$$

Закључак

У раду се описују и анализирају мере за оцену перформанси рачунара. Све више се схвата важност улоге коју произвођачи PC-а имају у остварењу напретка у пољу PC система. На основу захтева купаца, произвођачи праве побољшања у поузданости процесора, меморијског система и перформансама. Процесор координира рад целокупног рачунарског система и врши обраду података. Највећи произвођачи процесора су INTEL и AMD. Меморија рачунарског система служи за меморисање података и мери се у бајтовима. Постоје разни типови меморија, а најпознатија је RAM која се може читати, а писање података је такође омогућено. Један од највећих произвођача је Kingston и Transcend.

У самом раду су кроз примере анализиране перформансе процесора и меморијских система. Циљ рада је да купцима помогнемо да уз што нижу цену добију што боље перформансе рачунара.

Литература:

[1] William Stallings, *Организација и архитектура рачунара*, Пројекат у функцији перформанси, 2006.

[2] Andrew S. Tanenbaum, *Архитектура и организација рачунара*, 2005.

[3] Слободан Обрадовић, *Рачунари*, 2003.

[4] Јован Ђорђевић, *Архитектура и организација рачунарског система*, 2003.

[5] Јован Ђорђевић, *Приручник из архитектуре и организације рачунара*, 1999.

[6]https://sr.wikipedia.org/sr-ec/DLX_(%D0%B0%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D0%BE%D1%80%D0%B0) преузето 10.04.2020.

[7]https://en.wikibooks.org/wiki/Microprocessor_Design/Pipelined_Processorsпреу
преузето 18.05.2020.

[8]https://en.wikibooks.org/wiki/Microprocessor_Design/Pipelined_Processors
преузето 18.05.2020.

[9]https://sr.wikipedia.org/sr-ec/%D0%A1%D1%83%D0%BF%D0%B5%D1%80%D1%81%D0%BA%D0%B0%D0%BB%D0%B0%D1%80%D0%BD%D0%BE%D1%81%D1%82 преузето 25.05.2020.

[10]https://sr.wikipedia.org/wiki/%D0%95%D1%80%D0%B0%D1%82%D0%BE%D1%81%D1%82%D0%B5%D0%BD%D0%BE%D0%B2%D0%BE_%D1%81%D0%B8%D1%82%D0%BE преузето 26.05.2020.

[11] https://en.wikipedia.org/wiki/Dhrystone преузето 04.06.2020.

[12]https://sr.wikipedia.org/sr-ec/%D0%9A%D0%B5%D1%88_%D1%86%D0%B5%D0%BD%D1%82%D1%80%D0%B0%D0%BB%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D0%BE%D1%80%D1%81%D0%BA%D0%B5_%D1%98%D0%B5%D0%B4%D0%B8%D0%BD%D0%B8%D1%86%D0%B5 преузето 12.06.2020.

[13] https://en.wikipedia.org/wiki/Perpendicular_recording преузето 15.06.2020.

[14] https://en.wikipedia.org/wiki/Standard_RAID_levels преузето 16.06.2020.