- Performanse računara
  - Benchmark programi
    - Kompajleri često imaju flegove za pravljenje koda za benchmark programe koji uvode optimizacije koje bi drugde proizvele nekorektan kod ili usporenja
    - Da li se sme menjati source kod za različite sisteme?
      - Source se ne sme menjati
      - Source se može menjati, ali je praktično neizvodljivo (npr. testiranje proprietary programa)
      - Source se može menjati dok god daje isti izlaz
    - Korišćenje kolekcija benchmark programa (benchmark suite)
      - Uprosečuju eventualne negativne uticaje
    - Jedna od boljih standradizovanih benchmark kolekcija je SPEC (Standard Performance Evaluation Corporation), www.spec.org
    - Phoronix Test Suite (veoma skriptabilan)

- Performanse računara
  - Benchmark programi
    - SPEC za desktop računare
      - procesorski-intenzivni (integer/floating point)
      - grafički-intenzivni (OpenGL/DirectX/...)
    - SPEC za servere - puno različitih namena
      - Procesorska propusna moć
      - TLP - testovi zasnovani na OpenMP, MPI, OpenAAC, OpenCL
      - Testovi za fajl sisteme, baze podataka, Java, ...
    - Nema smisla objaviti rezultate merenja performansi ako se ne mogu reprodukovati
      - SPEC u rezultate uključuje i opis hardvera računara, softvera, felgova kompajlera, itd - svega potrebnog za reprodukciju testa
        - Slično i PTS (openbenchmarking.org)

- Performanse računara
  - Benchmark programi
    - Sumiranje rezultata
      - Postoji dosta načina za kombinovanje rezultata više testova u jednu vrednost
      - Za svaki pojedinačni test će se podeliti vreme izvršavanja referentnog sa vremenom izvršavanja testiranog računara (SPECRatio)
      - Za dobijene vrednosti će se izračunati geometrijska sredina

Benchmarks	Sun Ultra Enterprise 2 time (seconds)	AMD A10-6800K time (seconds)	SPEC 2006Cint ratio	Intel Xeon E5-2690 time (seconds)	SPEC 2006Cint ratio	AMD/Intel times (seconds)	Intel/AMD SPEC ratios
perlbench	9770	401	24.36	261	37.43	1.54	1.54
bzip2	9650	505	19.11	422	22.87	1.20	1.20
gcc	8050	490	16.43	227	35.46	2.16	2.16
mcf	9120	249	36.63	153	59.61	1.63	1.63
gobmk	10,490	418	25.10	382	27.46	1.09	1.09
hmmer	9330	182	51.26	120	77.75	1.52	1.52
sjeng	12,100	517	23.40	383	31.59	1.35	1.35
libquantum	20,720	84	246.08	3	7295.77	29.65	29.65
h264ref	22,130	611	36.22	425	52.07	1.44	1.44
omnetpp	6250	313	19.97	153	40.85	2.05	2.05
astar	7020	303	23.17	209	33.59	1.45	1.45
xalancbmk	6900	215	32.09	98	70.41	2.19	2.19
Geometric mean			31.91		63.72	2.00	2.00

- Principi kvantitativnog dizajna računara
  - Principi zasnovani na merenjima performansi i potrošnje energije
  - Princip lokalnosti
    - Programi često koriste podatke i kod koji su nedavno koristili
      - Generalno, oko 90% vremena se troši na oko 10% koda
        - Više važi za kod nego za podatke
    - Vremenska lokalnost - nedavno korištene naredbe će se verovatno uskoro opet koristiti
    - Prostorna lokalnost - adresa narednog podatka/naredbe će verovatno biti blizu prethodne

- Principi kvantitativnog dizajna računara
  - Amdalov zakon
    - Dobitak na ukupnim performansama uvođenjem nekog poboljšanja je limitiran udelom vremena u kome se unapređenje može koristiti u ukupnom vremenu izvršavanja
    - Ubrzanje nam određuje koliko brže će se zadatak izvršiti kada se koristi unapređenje u odnosu na varijantu bez tog unapređenja

$$Ubrzanje = \frac{Performanse\ sa\ unapređenjem}{Performanse\ bez\ unapređenja}$$

$$Ubrzanje = \frac{Vreme\ izvršavanja\ bez\ unapređenja}{Vreme\ izvršavanja\ sa\ unapređenjem}$$

- Principi kvantitativnog dizajna računara
  - Amdalov zakon
    - Ubrzanje zavisi od
      - Udela vremena izvršavanja originalnog zadatka koje se može izmeniti tako dakoristi unapređenje
        - npr. ako se program izvršava 100 sekundi, a od toga kod koji se izvršava u 40 sekundi ukupnog vremena se može unaprediti, onda je ovaj odnos 40/100=0.4
        - $\text{Udeo}_{\text{unapređeno}}$ - uvek manje ili jednako 1 (u idealnom slučaju)
      - Koliko brže se deo koji se može unaprediti izvršava u odnosu na originalni zadatak
        - npr. ako unapređenje izaziva da se kod koji se može unaprediti izvršava za 4 umesto za 40 sekundi, tada je ovaj odnos 40/4=10
        - $\text{Ubrzanje}_{\text{unapređeno}}$ - uvek veće od 1 (ako stvarno postoji)

$$Vreme\,izvršavanja_{novo} = Vreme\,izvršavanja_{staro} \cdot \left(\left(1 - Udeo_{unapređeno}\right) + \frac{Udeo_{unapređeno}}{Ubrzanje_{unapređeno}} \right)$$

$$Ubrzanje_{ukupno} = \frac{Vreme\,izvršavanja_{staro}}{Vreme\,izvršavanja_{novo}} = \frac{1}{\left(\left(1 - Udeo_{unapređeno}\right) + \frac{Udeo_{unapređeno}}{Ubrzanje_{unapređeno}} \right)}$$

- Principi kvantitativnog dizajna računara
  - Amdalov zakon
    - Dobra vodilja koja može reći koliko će neko unapređenje unaprediti ukupne performanse
      - I samim tim reći da li ima smisla trošiti resurse na razvoj unapređenja
    - Dobar način za poređenje različitih unapređenja, kako bi se utvrdilo gde je bolje alocirati razvojne resurse
    - Problem je što je nekad teško izmeriti $\text{Udeo}_{\text{unapređeno}}$

- Principi kvantitativnog dizajna računara
  - Jednačina performansi procesora
    - Generalno, svi procesori imaju konstantan klok (ako izuzememo mogućnost dinamičke izmene frekvencije)
    - Tada jedan ciklus (period kloka) ima konstantnu dužinu, koja se može izraziti ili preko vremena (npr. 1ns) ili preko frekvencije (npr. 1GHz)
    - Tada je procesorsko vreme potrebno za izvršavanje programa:

$$CPU_{vreme} = CPU_{broj\ ciklusa\ programa} \cdot Trajanje\ ciklusa = \frac{CPU_{broj\ ciklusa\ programa}}{Frekvencija}$$

- Principi kvantitativnog dizajna računara
  - Jednačina performansi procesora
    - Pored broja ciklusa, može se izbrojati i koliko naredbi je izvršeno (IC - instruction count)
    - Ako se zna i ukupan broj cikluca, može se odrediti i srednji broj ciklusa po naredbi (CPI - clock cycles per instruction):

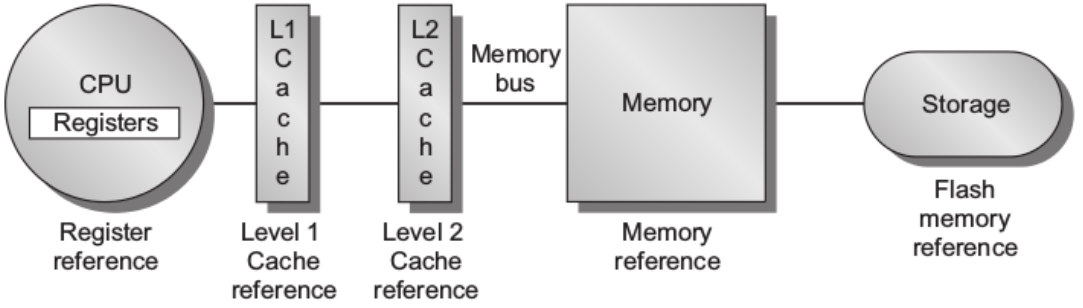
$$CPI = \frac{CPU_{broj\ ciklusa\ programa}}{Broj\ naredbi}$$

    - Sada je procesorsko vreme:

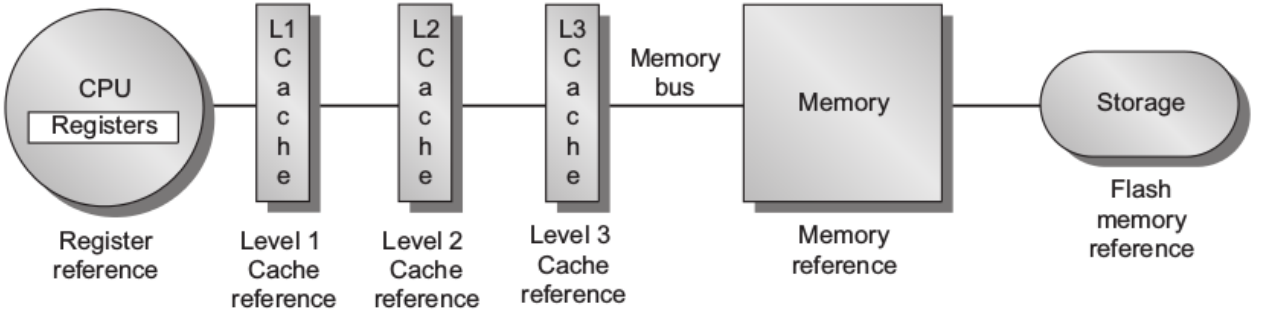
$$CPU_{vreme} = Broj\ naredbi\ programa \cdot CPI \cdot Trajanje\ ciklusa$$

    - Pa stoga, performanse procesora zavise od
      - frekvencije rada (dužine ciklusa)
      - srednjeg broja ciklusa po naredbi
      - broja naredbi u zadatku
    - Zavisnost je ista od sva 3 parametra, ali oni međusobno nisu nezavisni

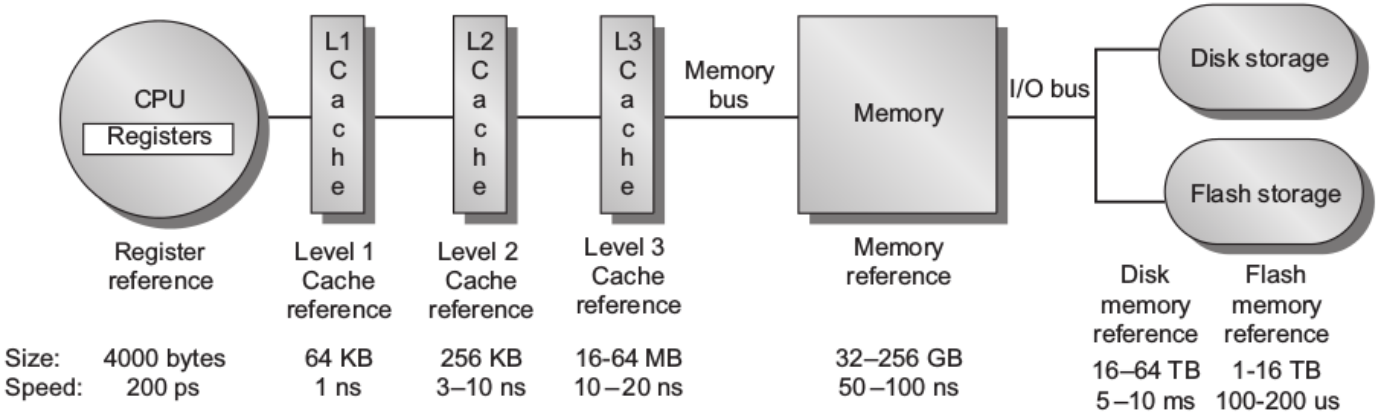
- Memorijska hijerarhija
  - Idealno bi bilo kada bi memorija bila beskonačna
  - Ekonomski prihvatljivo je da imamo memorijsku hijerarhiju
    - Princip lokalnosti
    - Odnos cene i performansi (kapaciteta) za različite memorijske tehnologije
  - Brža memorija je generalno skuplja, što se odražava na nivoe hijerarhije
    - Brže i skuplje memorije su manjeg kapaciteta i bliže vrhu hijerarhije



(A) Memory hierarchy for a personal mobile device

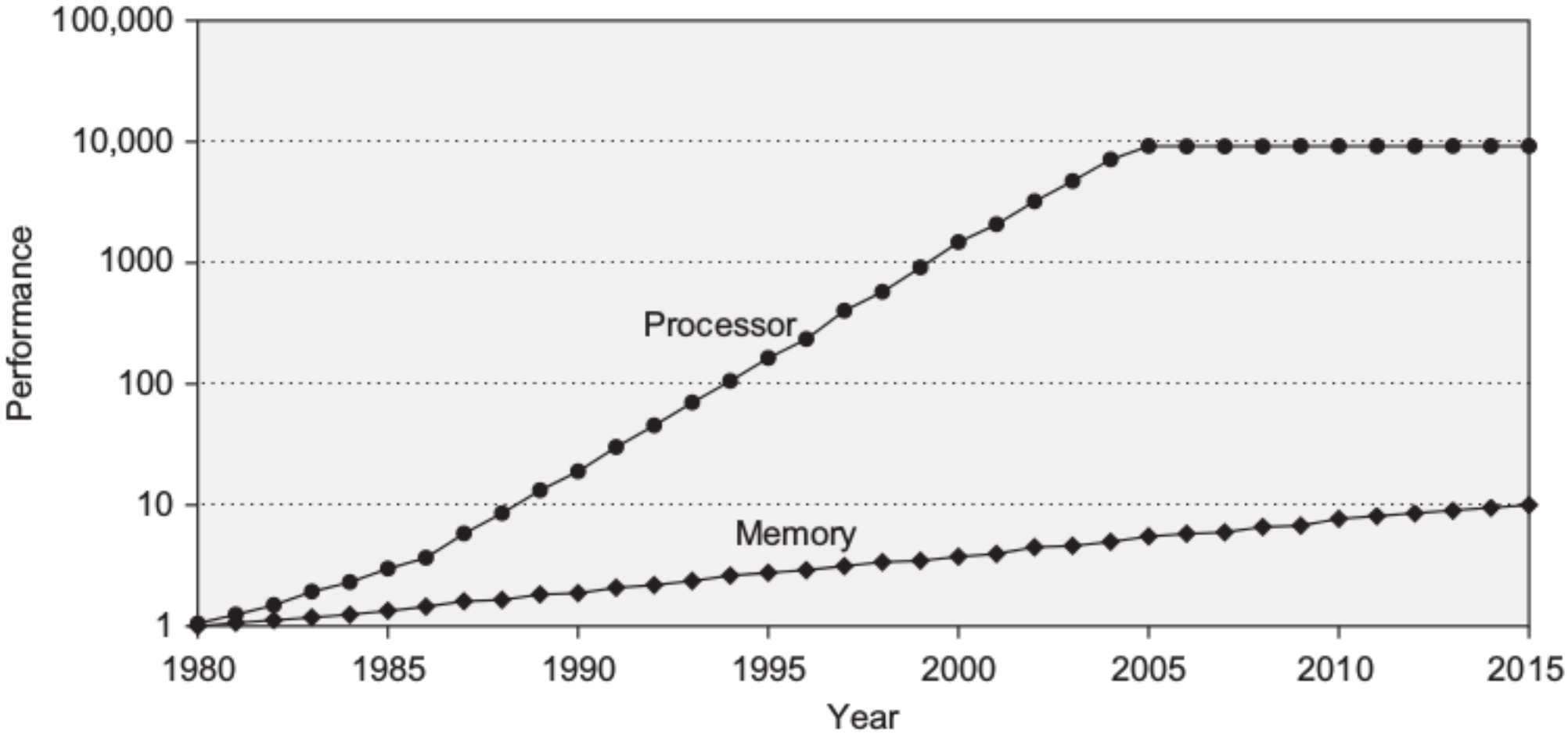


(B) Memory hierarchy for a laptop or a desktop



(C) Memory hierarchy for server

- Memorijska hijerarhija
  - Razlika relativnog porasta broja pristupa memoriji u sekundi od strane jednog jezgra procesora i broja pristupa u sekundi za jedan modul DRAM memorije u odnosu na stanje u 1980. godini



- Memorijska hijerarhija
  - Stvari se dodatno komplikuju kada se uvede više jezgara (traže veći protok), više čipova po DRAM modulu i više kanala ka DRAM modulima
  - Primer: Intel Core i7 6700, može da generiše dva pristupa memoriji po jezgru po ciklusu, što kombinovano sa pristupom memoriji za dobavljanje naredbi daje maksimalni protok od ~410GiB/s, dok je maksimalan protok DRAM-a na dva kanala ~34GiB/s (oko 8% onoga što procesor može da zatraži)
  - Optimizacija memorijske hijerarhije se donedavno isključivo bavila srednjim vremenom pristupa, ali je u poslednje vreme i količina energije postala bitan faktor (statička i dinamička snaga neophodna sa SRAM u keš memoriji, pristup DRAM-u)
  - Još veći problem kod prenosnih uređaja (keš može da troši 25-50% energije sistema)

- Memorijska hijerarhija
  - Keš memorija
    - Za 1 nivo: SRAM memorija koja se nalazi direktno u procesoru, koja je mnogo brža od DRAM-a i u koju se kopira sadržaj DRAM-a, radi bržeg pristupa
    - Kopira se uvek jedan blok memorije (obično nazvan linija)
      - Linija sadrži i oznaku koje memorijske adrese pokriva
    - Keš je obično asocijativna memorija
      - daje mu se adresa na upit, a onda se paralelno proveravaju linije koje su u kešu, radi provere da li se tražena adresa poklapa sa njihovom oznakom
    - Keširanje za čitanje je jednostavno
    - Keširanje za upis
      - write-through - upis i u keš i u glavnu memoriju
      - write-back - upis samo u keš, a u glavnu memoriju kada se linija izbacuje
      - obe strategije mogu da kotiste write buffer-e kako bi se smanjilo čekanje na glavnu memoriju

- Memorijska hijerarhija
  - Keš memorija
    - Jedna od mera efikasnosti keš memorije je procenat promašaja
      - procenat pristupa kešu kada se u njemu ne nalazi traženi sadržaj
    - Kategorije promašaja:
      - obavezni - dešava se kod prvog pristupa nekoj adresi
      - zavisan od kapaciteta - kada keš ne može da sadrži sve linije potrebne za izvršavanje, pa se neke linije moraju osloboditi kako bi se ubacile nove
      - zavisan od konflikta u setu - ako se više linija mapira na isti set linija u kešu
    - Multithreading i više jezgara dodatno komplikuju priču o kešu
      - više procesa na raznim mestima u memoriji se takmiči za prostor u kešu
      - četvrta kategorija promašaja: zbog koherentnosti (kada više jezgara keširaju istu liniju memorije)

- Memorijska hijerarhija
  - Keš memorija
    - Još jedna mera efikasnosti su promašaji po naredbi (umesto po memorijskom pristupu)

$$Promašaji_{\,po\,naredbi} = \frac{Frekvencija\ promašaja \cdot Broj\ pristupa\ memoriji}{Broj\ naredbi} = Frekvencija\ promašaja \cdot Broj\ pristupa\ memoriji_{\,po\,naredbi}$$

    - Nijedna od ove dve mere ne uračunava koštanje promašaja, pa stoga imamo i srednje vreme pristupa memoriji:

$$Srednje\ vreme\ pristupa = Vreme\ pogotka + Frekvencija\ promašaja \cdot Vreme\ promašaja$$

      gde je vreme pogotka vreme potrebno za čitanje podatka iz keša, a vreme promašaja vreme potrebno da se podaci prebace iz glavne memorije u keš

- Memorijska hijerarhija
  - Keš memorija
    - Osnovne tehnike optimizacije
      - Veća veličina linije - bolje korišćenje prostorne lokalnosti, ali i duže vreme kod promašaja; za mali keš može i povećati broj promašaja
      - Veći keš - potencijalno duže vreme da se nađe linija kod pogotka, veća potrošnja energije
      - Veća asocijativnost - dolazi i sa većim vremenom za pretragu kod pogotka, i sa većom potrošnjom energije
      - Keš u više nivoa - svaki sledeći nivo je veći i sporiji
      - Davanje prioriteta promašajima u čitanju u odnosu na promašaje u pisanju (uz korišćenje write buffer-a)
      - Izbegavanje pretvaranja u fizičku adresu kada postoji virtuelna memorija - indeksiranje linija keša sa adresama unutar stranice memorije (one su identične za za stranicu koja je fizički u memoriji i onu koja je na disku)