Keširanje

Literatura: Hennessy & Patterson, B.1–B.2

Rekapitulacija za memoriju/procesor

- Inverzna korelacija složenosti memorije i njenih performansi
- Praktičan sistem mora imati veliku količinu memorije da bi mogao da obrađuje realne skupove podataka
  - dakle, radna memorija mora biti sporija u odnosu na procesor
- Skupovi podataka su često veći i od praktično upotrebljive količine radne memorije
  - sekundarna memorija je nekoliko redova veličine sporija od glavne
  - ovo se donekle menja postojanjem bržih sekundarnih memorija (flash)

Moguće arhitekture međumemorije

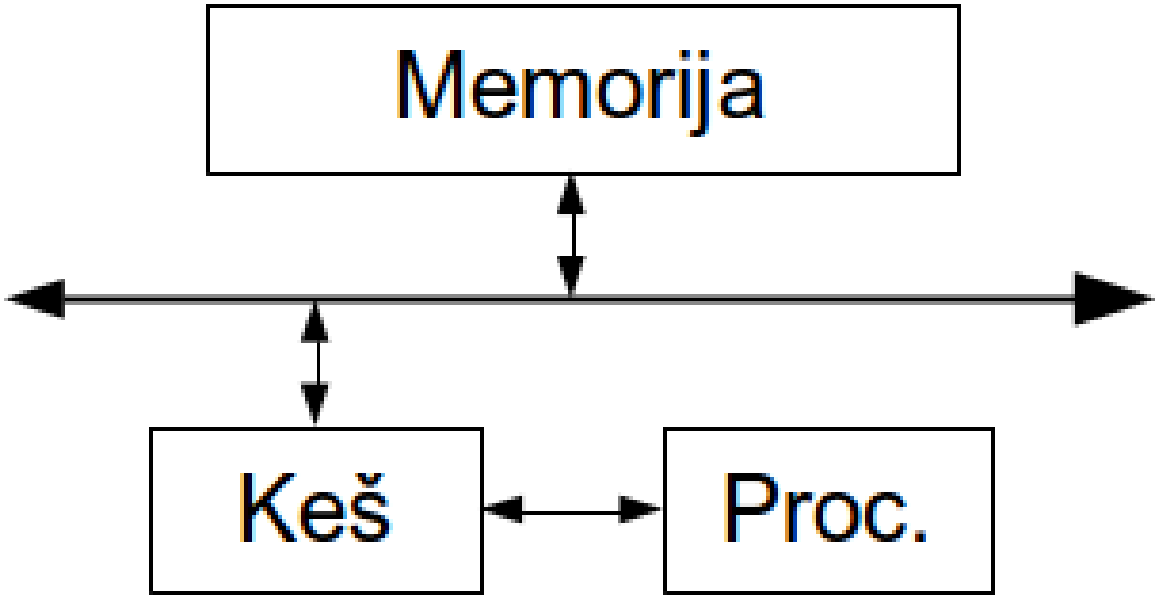
- Uvodi se mala količina brze međumemorije
- Jedna od mogućnosti je da se memorija tretira kao prošireni skup registara pod kontrolom korisničkih procesa
  - procesi moraju znati hardverske detalje pojedinačnih procesora
  - svaki proces mora imati disjunktni podskup u odnosu na druge
  - trošak organizovanja pristupa bi poništio prednosti
- Druga mogućnost, koja se univerzalno koristi, jeste međumemorija (uglavnom) transparentna za korisnika

Keš memorija: principi

- Brza memorija koja sadrži privremene kopije podataka za koje se može pretpostaviti da će biti potrebni za izvršavanje koda
- Ovo je izvodljivo jer programi ispoljavaju *lokalitet*
- Prostorni lokalitet: mali kontinualni podskup podataka se koristi u kontinuitetu
  - na primeru koda: petlja
  - izvesne konfiguracije podataka su takođe prostorno bliske
- Vremenski lokalitet: verovatnoća da će neki podskup podataka biti opet potreban ne dugo posle obrade
  - za kod: funkcijski poziv u petlji (kod funkcije je prostorno udaljen, ali potreban kod svake iteracije)
  - za podatke: ukupan radni skup relativno ograničen, ali ne mora biti prostorno kontinualan

Konfiguracija keša (1)

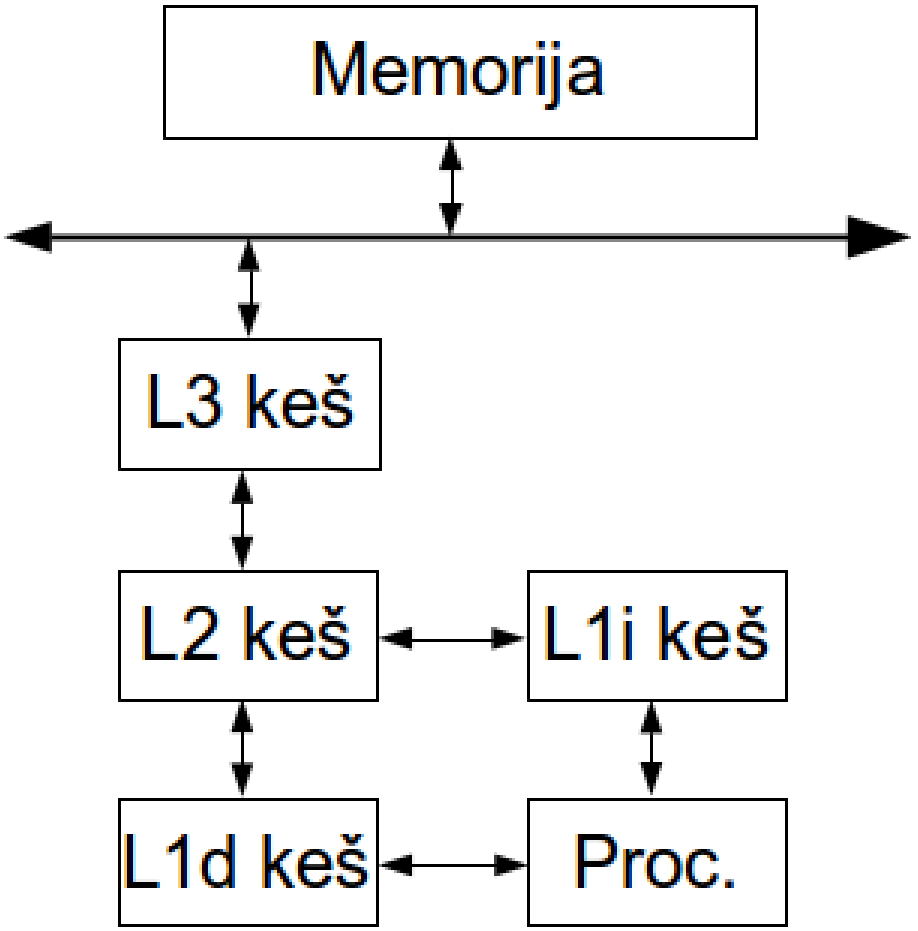
- Prvobitni sistemi:



Konfiguracija keša (2)

- Sav pristup memoriji ide kroz keš
- Veza između keša i procesora je specijalno projektovana za brzinu
- Ovo funkcioniše, ali moguća su poboljšanja:
  - razdvajanje memorije za instrukcije i podatke: modifikacija klasične fon Nojmanove arhitekture
  - organizacija u nivoima, zbog neekonomičnosti povećanja jedinstvenog memorijskog prostora

Moderna konfiguracija keša



Osnovni parametri keša (1)

Vrsta pristupa	Br. ciklusa	Vreme ns
L1 pogodak	4	1,2
L2 pogodak	10	3,0
L3 pogodak	40	12,0
lokalni RAM	100	60
udaljeni RAM	160	100

Osnovni parametri keša (2)

- Pretpostavka: 100 elemenata, 100 iteracija
- Svaki pristup kroz lokalni RAM: 100·100·100 = 1.000.000 ciklusa
- Svaki pristup kroz L2 keš: 100·100·10 = 100.000 ciklusa
- Ušteda: 90%
- U realnim situacijama iskorišćenja iznad 90% su uobičajena

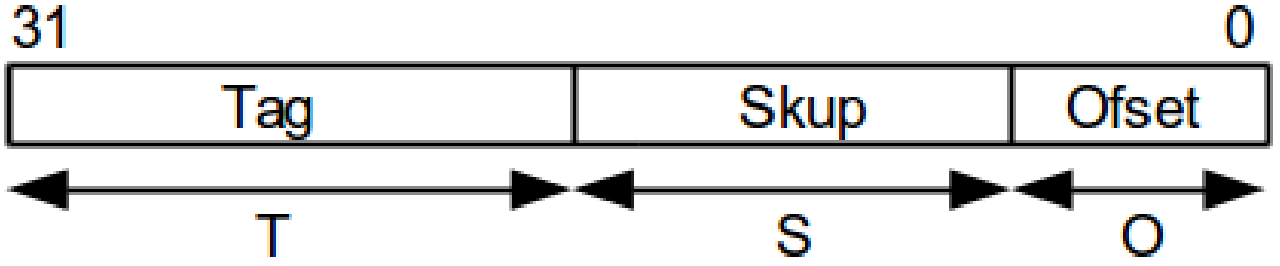
Osnovna organizacija keša (1)

- Tipično, keš je hiljadama puta manji od količine RAM-a
- Potrebno je naći efikasan način za popunjavanje i adresiranje sadržaja
- Svaki zapis u kešu mora biti dostupan na osnovu adrese podatka koji se traži
  - deo zapisa koji predstavlja adresu zove se **tag**
- Granularnost zapisa na nivou mašinske reči bila bi krajnje neefikasna
  - zato se postavlja na 32 ili 64 bajta, što predstavlja jednu **liniju**

Osnovna organizacija keša (2)

- Prilikom izmene memorije, cela linija prvo mora da se učita u keš
  - u principu, procesor nema operacije koje rade nad celom linijom
- Izmena neke lokacije u liniji označava celu liniju kao *zaprljanu*
  - zaprljane linije su one koje su izmenjene, a nisu zapisane nazad u memoriju
- Prilikom čitanja, najčešće se neka od postojećih linija mora izbaciti iz keša
  - ako su svi kandidadi zaprljani, moraju se zapisati, što unosi kašnjenje
  - izbor kandidata je generalno težak problem
- Za primer: 4 MB, 64 bajta, 65536 linija (22 bita)

Adresiranje linija



- Segment O je fiksan i zavisi od dužine linije: O = log2L
- Segmenti S i T zavise od načina organizacije adresiranja
  - potpuno asocijativno
  - direktno mapirano
  - skupovno asocijativno

Potpuno asocijativni keš

- S = 0, T = 32 - O
- Svaka linija može sadržati kopiju bilo koje memorijske lokacije
  - obratite pažnju: i dalje postoji mogućnost promašaja
- Za svaku od linija mora postojati komparator koji će proveriti jednakost tagova
- Hardverski preterano zahtevno
- Komparatori ne mogu da rade iterativno jer se inače gubi smisao keširanja

Direktno mapirani keš

- Hardverski problem se rešava ograničavanjem opsega pretrage
- Na jednoj krajnosti, svaki tag se mapira na tačno jednu liniju
- $S = \log_2 N$, gde je N broj linija
  - u našem slučaju N = 65536, S = 16
- Može dobro da radi ako su adrese ravnomerno raspoređene u prostoru određenom za mapiranje
  - najčešće nisu
  - praktična posledica: neke linije se stalno zamenjuju, neke ostaju prazne

Skupovno asocijativni keš

- Kombinacija prethodna dva
- **S** više ne bira pojedinačne linije, već skupove linija
  - slično direktno mapiranom, ali je promenjena granularnost adresiranja
- Unutar svakog skupa, linije koje mu pripadaju porede se s tagom u paraleli
  - slično potpuno asocijativnom, samo za mali broj elemenata
- Na našem primeru, ako je keš 8-struko asocijativan, imaće 8192 skupa
  - $8192 \cdot 8 = 65536$
  - $S = \log_2 8192 = 13$

Primer—direktno mapiran keš

- Polazna memorijska adresa: 0x9cb2008
- Binarno, sa grupisanim ciframa:
  1001 1100 1011 0010 0000 0000 1000
- Ista ta vrednost, sa razdvojenim tagom i selektorom linije unutar keša:
  1001 1100 1011 0010 0000 0000 1000
  tag: 0x27, selektor: 0x2c80, ofset: 0x8
- Adresa: 0xa0b2008, isti postupak:
  1010 0000 1011 0010 0000 0000 1000
  tag: 0x28, selektor: 0x2c80, ofset: 0x8
- Ako program naizmenično pristupa ovim adresama, linija će svaki put morati da se obnavlja iz RAM-a

Primer—8-struko asocijativni keš

- Polazna memorijska adresa: 0x9cb2008, isti postupak kao i ranije, ali ovog puta će selektor biti 13 bita umesto 16:
  1001 1100 1011 0010 0000 0000 1000
  tag: 0x139, selektor: 0xc80, ofset: 0x8
- Polazna adresa 0xa0b2008:
  1010 0000 1011 0010 0000 0000 1000
  tag: 0x141, selektor: 0xc80, ofset: 0x8
- Obe adrese se mapiraju na isti skup, ali imaju različite tagove, pa pošto skup ima kapacitet od osam linija, veće su šanse da za obe linije ima mesta u kešu

Keširanje instrukcija

- Jednostavnije od keširanja podataka
- Programski prevodioci generišu kod koji bolje vodi računa o iskorišćenju keša
  - autori prevodilaca generalno bolje poznaju ponašanje procesora
- Tok izvršavanja programa je predvidljiviji od šema pristupa podacima
- Programi po pravilu imaju dobar prostorni i vremenski lokalitet
- Eventualni problem: samomodifikujući kod