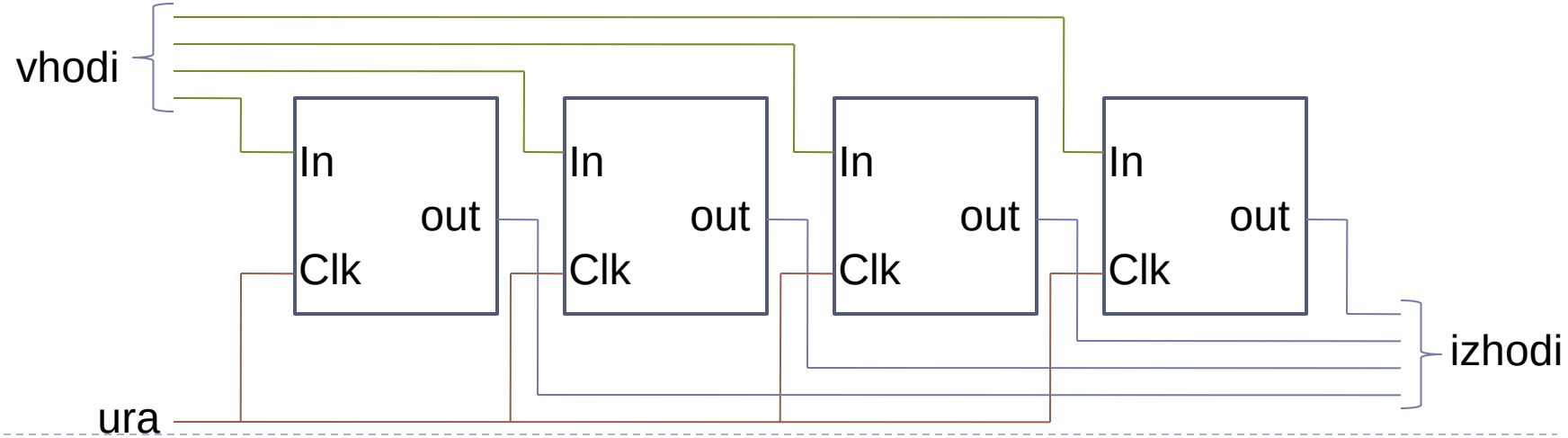
# Procesorski sistemi v telekomunikacijah
# Zgradba mikroprocesorja

# Register

- Pomnilnik, ki je vgrajen v mikroprocesor.
- Hrani podatek, ki je potreben za izvajanje neke operacije,
  ali pa je nastal kot rezultat opravljene operacije
- Zelo hitri – delujejo s frekvenco ure mikroprocesorja.
- Nekateri so dostopni programerju (npr. akumulatorji)
  drugi dostopni samo posredno (npr. programski števec)
  tretji pa sploh ne (npr. register, ki hrani pravkar prebran ukaz).
- Izvedba: skupina flip-flopov (1-bitnih pomnilniških celic)



# Akumulatorji in sorodni registri

- Akumulatorji
  - hranijo vhodne podatke za računske operacije.
  - v njih se shrani rezultat računske operacije.

  Primer: 68HC11 – dva 8-bitna akumulatorja (A in B)
  Primer: ARM7 – 16 splošnih 32-bitnih registrov (R0-R15)

- Indeksni registri

  Ponavadi namenjeni izračunu naslova v pomnilniku.
  Z njimi so možne le določene operacije (običajno +, -)
  Primer: 16-bitna registra X in Y v 68HC11
  Primer: 14-bitni registri I0-I7 v procesorju ADSP2181

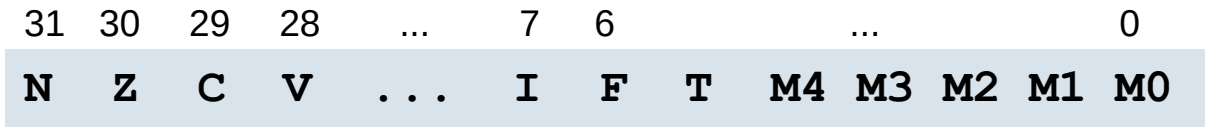
# Register stanja

- Biti v tem registru podajajo stanje mikroprocesorja
- Primer: register stanja v 68HC11


8 bitov v registru stanja

| | |
|---|---|
| N=1 | rezultat zadnje operacije je negativen (MSB=1) |
| Z=1 | rezultat zadnje operacije je nič |
| V=1 | zadnja operacija je povzročila preliv |
| C=1 | zadnja operacija je povzročila prenos |

- Primer: register stanja (CPSR) v ARM7TDMI jedru




32-bitov

- Register stanja ni nujno neposredno dostopen programerju.

# Programski števec

- Program counter (PC)
- Hrani pomnilniški naslov, od koder se bo prebrala naslednja beseda ukaza.
- Mikroprocesorji s podporo za hkratno večnitnost (SMT) imajo več programskih števcev.
- Pri cevovodnih mikroprocesorjih je vrednost PC ob izvajanju ukaza večja od naslova naslednjega ukaza.

ARM7 – en ukaz = 32 bitov = 4 celice = 4 zaporedni naslovi

| Ukaz | | | | | | |
|---|---|---|---|---|---|---|
| 1 | naloži 1 | dekodiraj 1 | izvedi 1 | | | |
| 2 | | naloži 2 | dekodiraj 2 | izvedi 2 | | |
| 3 | | | naloži 3 | dekodiraj 3 | izvedi 3 | |
| … | | | | … | … | … |
| | | | | | | |
| Urin cikel | 1 | 2 | 3 | 4 | 5 | … |
| PC | x | x+4 | x+8 | x+12 | x+16 | … |

# Sklad (Stack)

- Podatkovna struktura za podatke fiksne širine
  Npr. LPC2138 – 32 bitov, 68HC11 – 8 bitov
- Lahko se hrani v pomnilniku (npr. 68HC11, LPC2138)
  ali pa kar v samem mikroprocesorju (PICxxx)
- Operacija postavljanja na sklad: PUSH
- Operacija jemanja s sklada: POP
- Jemljemo v obratnem vrstnem redu, kot dajemo na sklad.
  Sklad je LIFO struktura (Last In, First Out)
- Velikost sklada (globina, depth) je omejena.
- Položaj (naslov) zadnjega postavljenega elementa v skladu kaže kazalec sklada (Stack Pointer, SP)

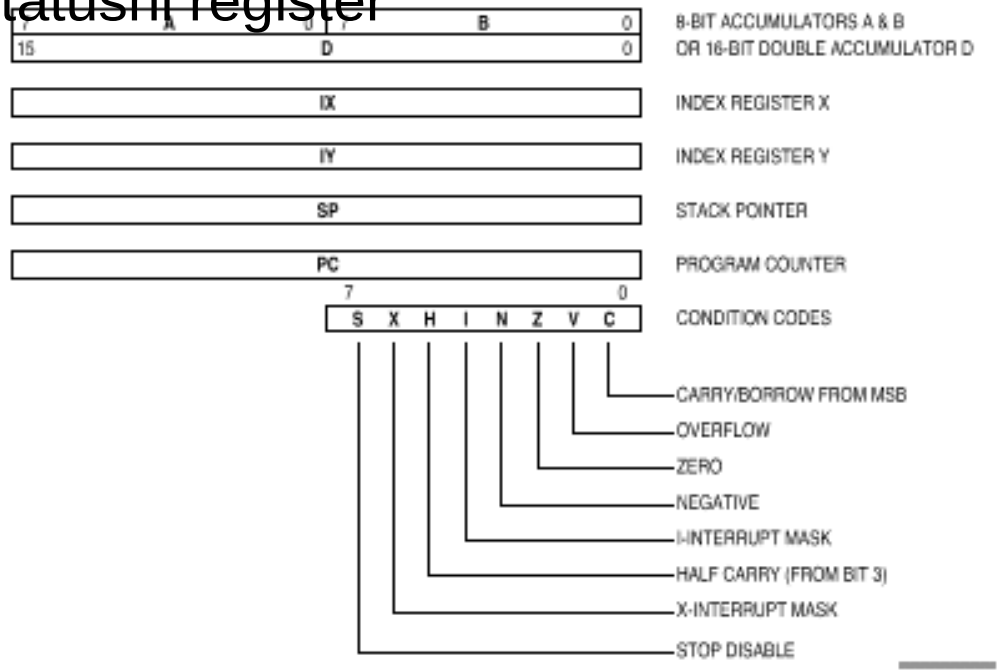
# Uporaba sklada

- Deluje kot garderoba – ko gremo notri, oddamo obleko (podatke), ko gremo ven, jo vzamemo v obratnem vrstnem redu, kot smo jo oddali.

- Za začasno shranjevanje vsebine registrov med izvajanjem podprogramov (t.j. funkcij v jeziku C)
- Za podajanje parametrov podprogramom
- Za hranjenje naslova, s katerega je bil poklican podprogram (da se lahko vrnemo nazaj, na mesto klica).

# Registri 68HC11

- Dva 8-bitna akumulatorja A in B (skupaj 16-biten akumulator D)
- Dva 16-bitna indeksna registra IX in IY
- 16-bitni programski števec (PC) in kazalec sklada (SP)
- 8-bitni statusni register



# Registri ARM7 …

- Registri so 32-bitni. Vsi (R0-R15) se lahko uporabljajo kot akumulatorji.
- Uporabniški način (User) in privilegirani načini delovanja.
- V uporabniškem (User) in privilegiranem sistemskem (System) načinu
  R0-R15 ... 32-bitni registri
  R15 = PC (programski števec)
  R14 = LR (Link Register, shrani vrednost PC pred skokom v podprogram)
  R13 = SP
  CPSR (Current Program Status Register) = statusni register (32-biten)
- Poleg sistemskega načina (System) so privilegirani načini še FIQ, Supervisor (po resetu, oz SWI prekinitvi), Abort, IRQ, Undefined
  Vsak od teh načinov ima
  - svoj R13 in R14 (svoj LR in SP)
  - svoj SPSR (Saved Program Status Register)
    (shrani vrednost CPSR pred skokom v privilegiran način)
- Način FIQ ima poleg tega še svoje verzije registrov R8-R12

# … registri ARM7

| System and User | FIQ | Supervisor | Abort | IRQ | Undefined |
|---|---|---|---|---|---|
| r0 | r0 | r0 | r0 | r0 | r0 |
| r1 | r1 | r1 | r1 | r1 | r1 |
| r2 | r2 | r2 | r2 | r2 | r2 |
| r3 | r3 | r3 | r3 | r3 | r3 |
| r4 | r4 | r4 | r4 | r4 | r4 |
| r5 | r5 | r5 | r5 | r5 | r5 |
| r6 | r6 | r6 | r6 | r6 | r6 |
| r7 | r7 | r7 | r7 | r7 | r7 |
| r8 | r8_fiq | r8 | r8 | r8 | r8 |
| r9 | r9_fiq | r9 | r9 | r9 | r9 |
| r10 | r10_fiq | r10 | r10 | r10 | r10 |
| r11 | r11_fiq | r11 | r11 | r11 | r11 |
| r12 | r12_fiq | r12 | r12 | r12 | r12 |
| r13 | r13_fiq | r13_svc | r13_abt | r13_irq | r13_und |
| r14 | r14_fiq | r14_svc | r14_abt | r14_irq | r14_und |
| r15 (PC) | r15 (PC) | r15 (PC) | r15 (PC) | r15 (PC) | r15 (PC) |

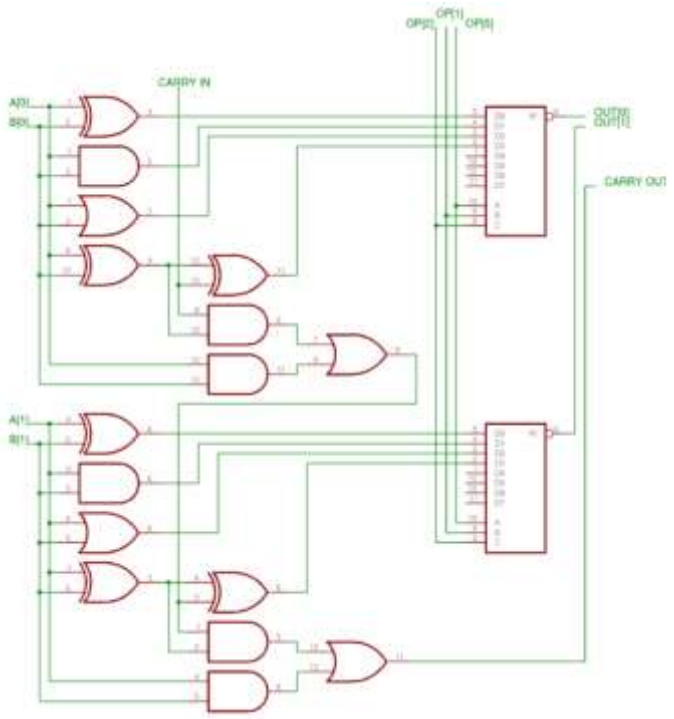
**ARM state program status registers**

| | | | | | |
|---|---|---|---|---|---|
| CPSR | CPSR | CPSR | CPSR | CPSR | CPSR |
| | SPSR_fiq | SPSR_svc | SPSR_abt | SPSR_irq | SPSR_und |

= *banked register*

# Aritmetično-logična enota (Arithmetic Logic Unit - ALU)

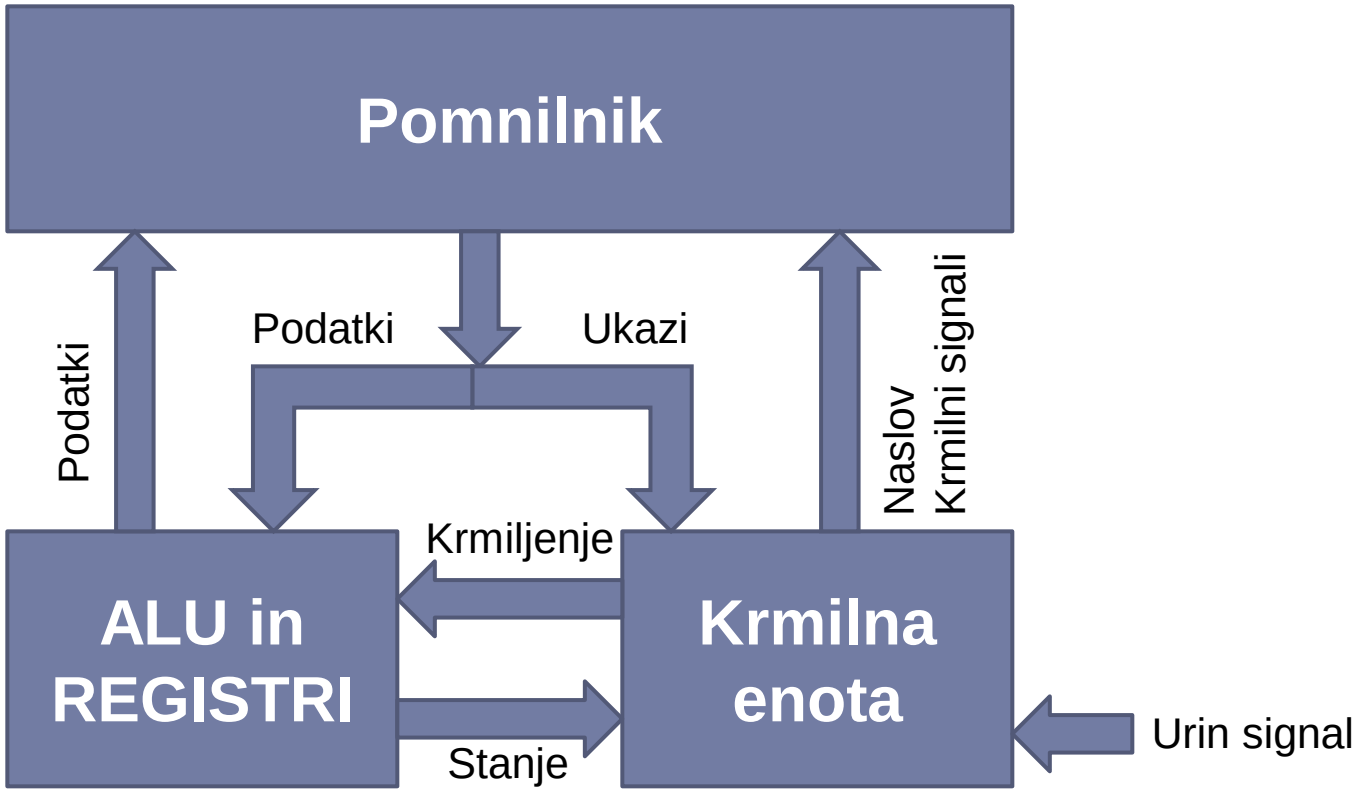
- Vezje za opravljanje operacij
  - aritmetičnih (+, -, včasih tudi * in /)
  - logičnih (AND, OR, EXOR, NOT)
  - pomikov in rotacij bitov
- Običajno izvajajo operacije na celih številih, ki morajo biti kodirana v obliki zapisa z dvojiškim
  n.




2-bitni ALU, operacije AND, OR, EXOR, +


8-bitni ALU, Texas Instruments SN74AS888

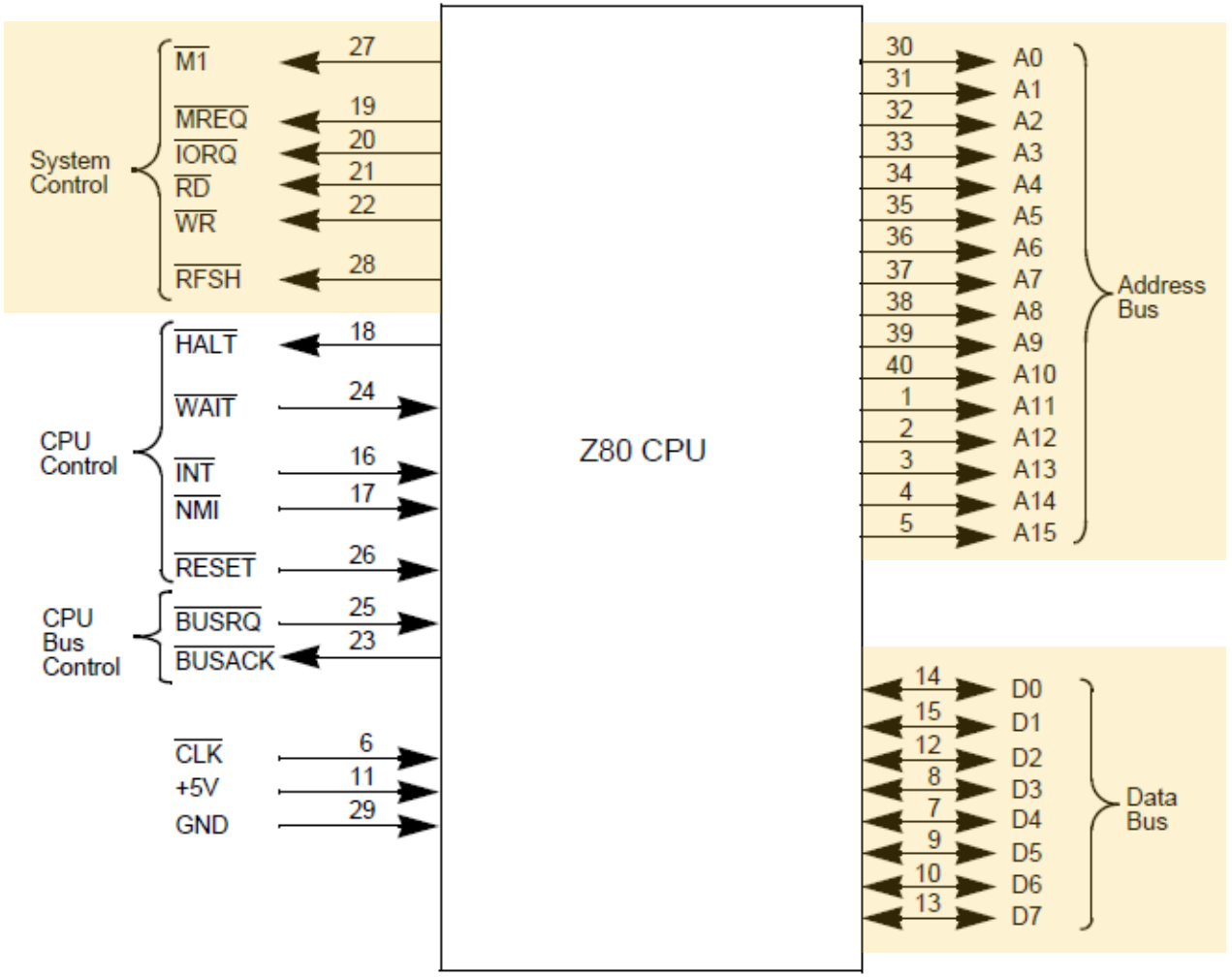
# von Neumannova arhitektura




- Čedalje več pomnilnika in čedalje hitrejši procesorji
  Hitrost povezave do pomnilnika ne raste enako hitro.
  **von Neumannovo ozko grlo**

# von Neumannova arhitektura
# Primer: Zilog Z80

krmiljenje

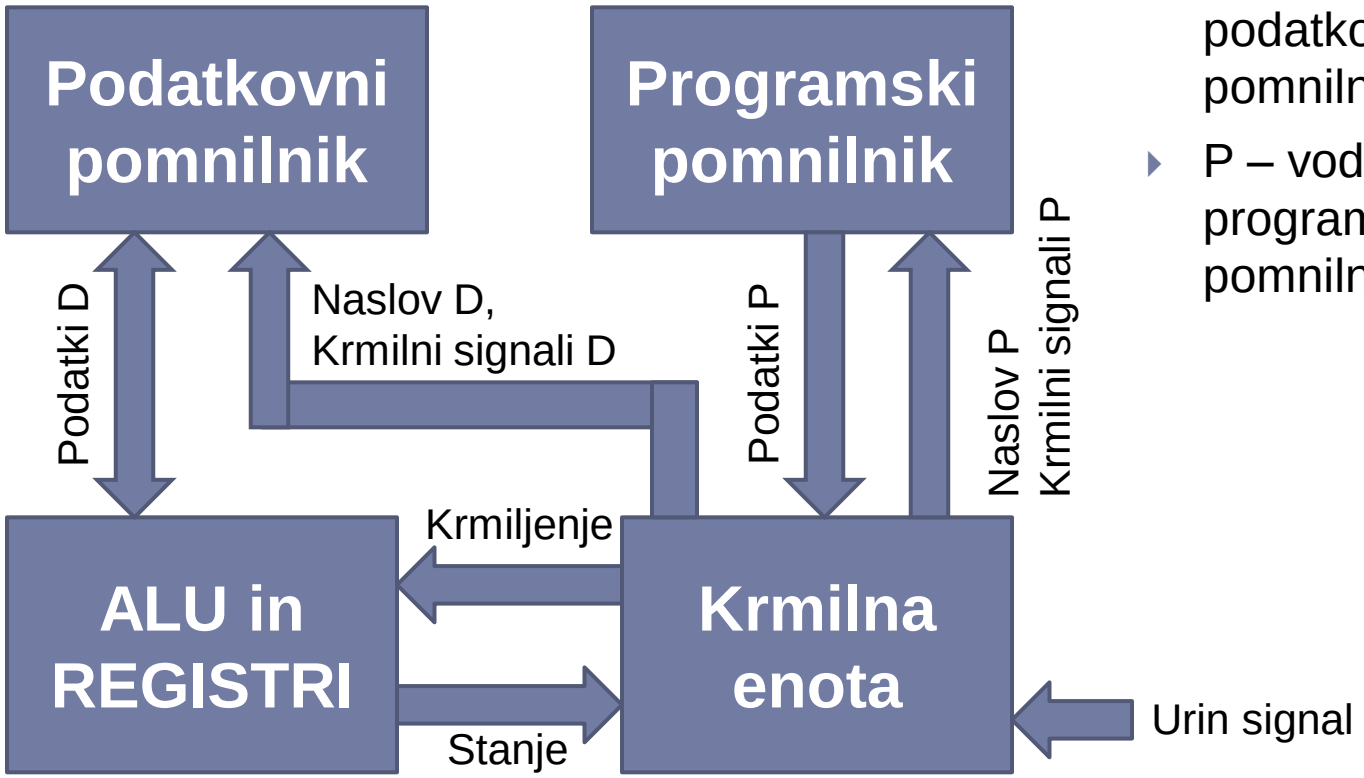



naslov

podatki

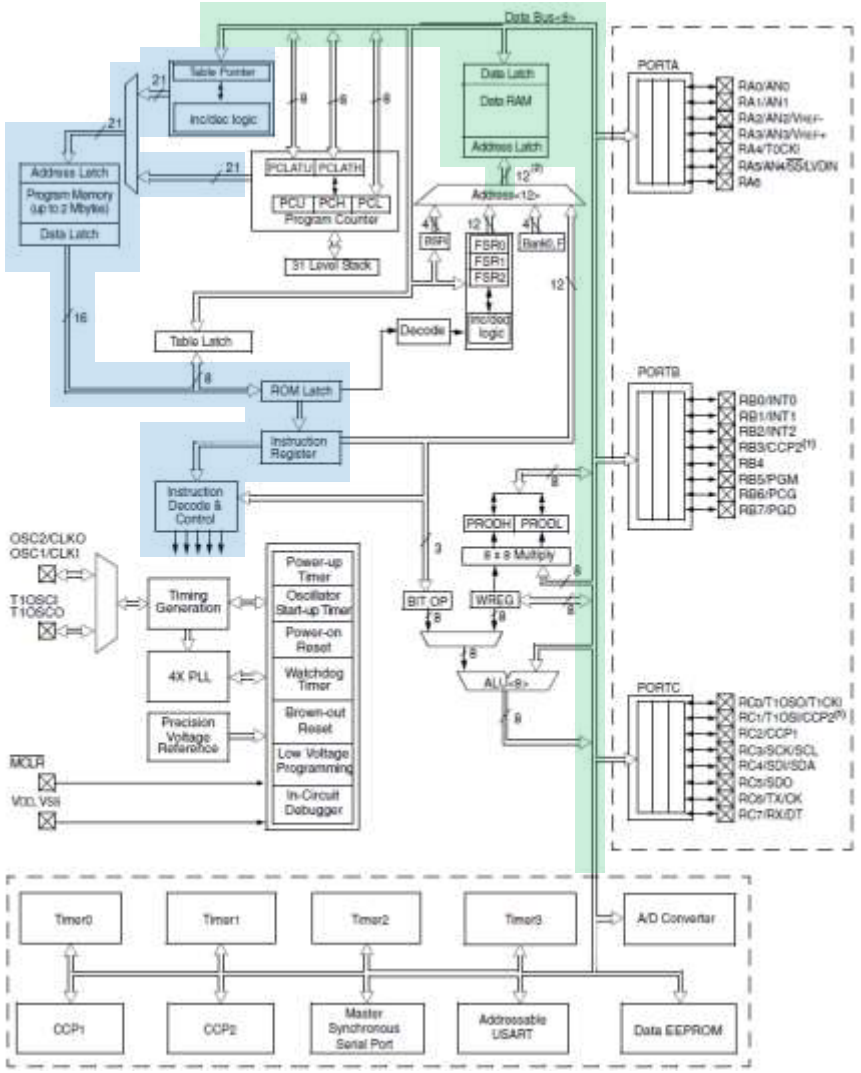
# Harvardska arhitektura

- Dva ločena naslovna prostora (program in podatki)
- Programski pomnilnik samo beremo

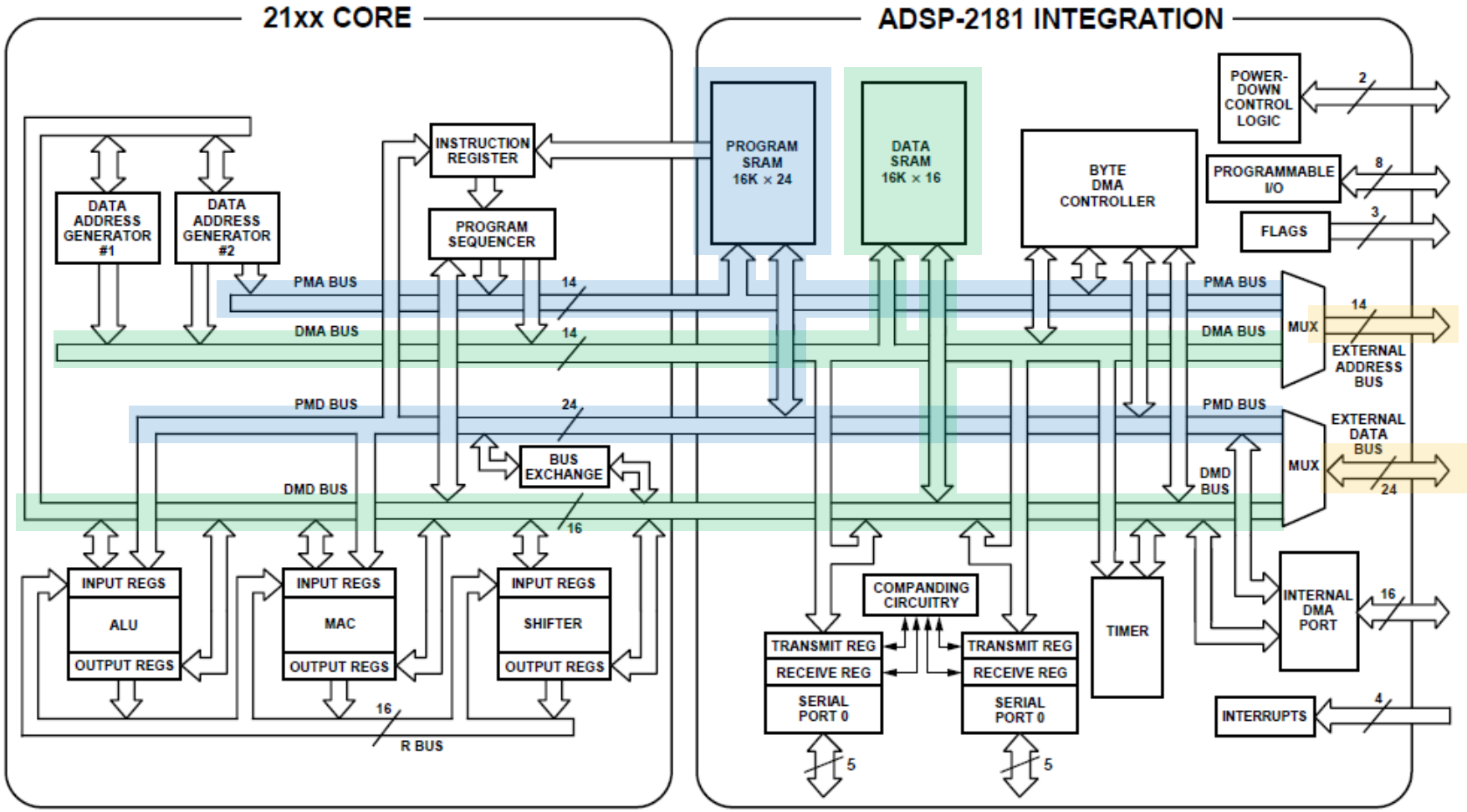



- D – vodilo podatkovnega pomnilnika
- P – vodilo programskega pomnilnika

# Harvardska arhitektura
# Primer: PIC18F2X2

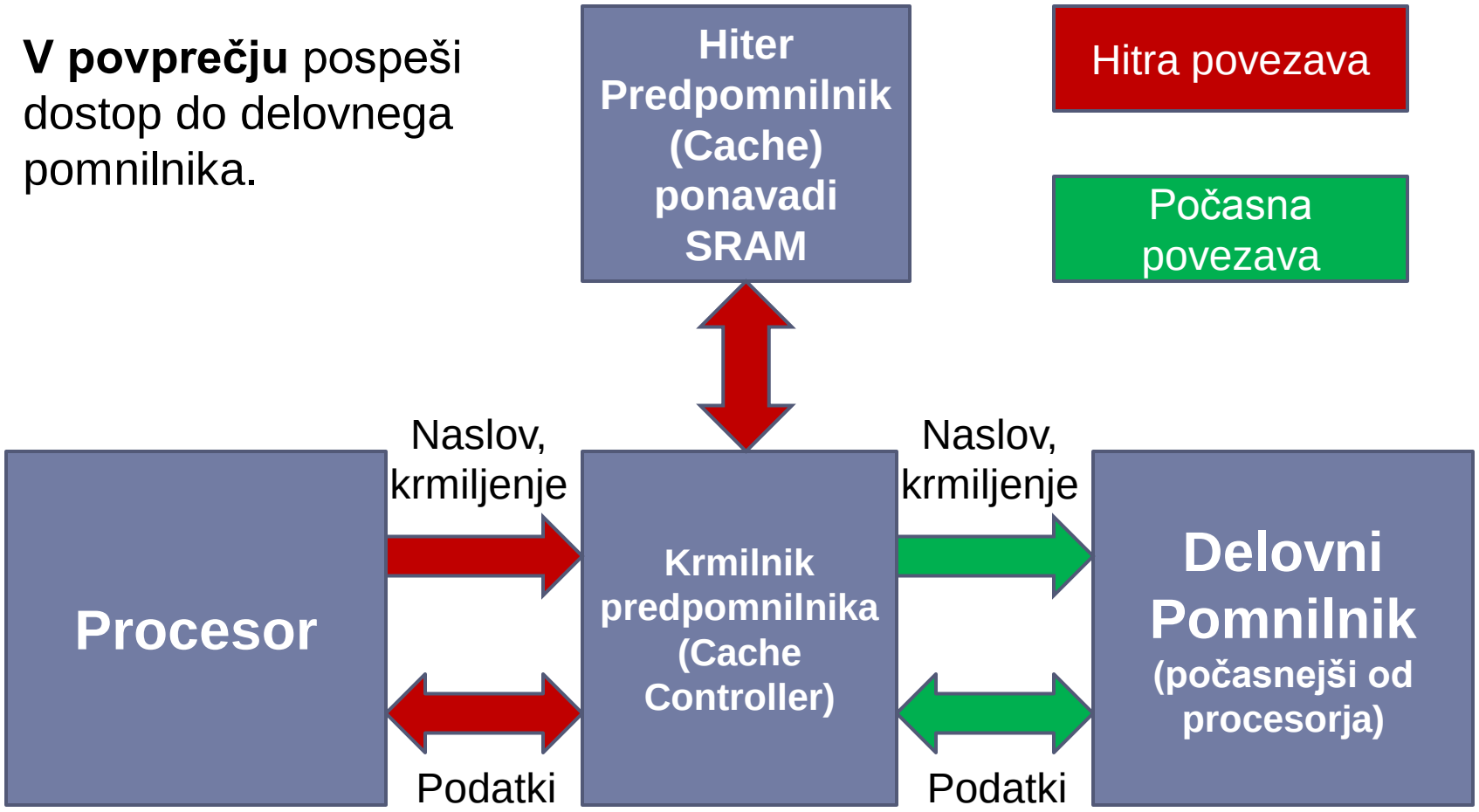


# (ne čisto) Harvardska arhitektura
# Primer: ADSP2181



# Predpomnilnik (Cache)

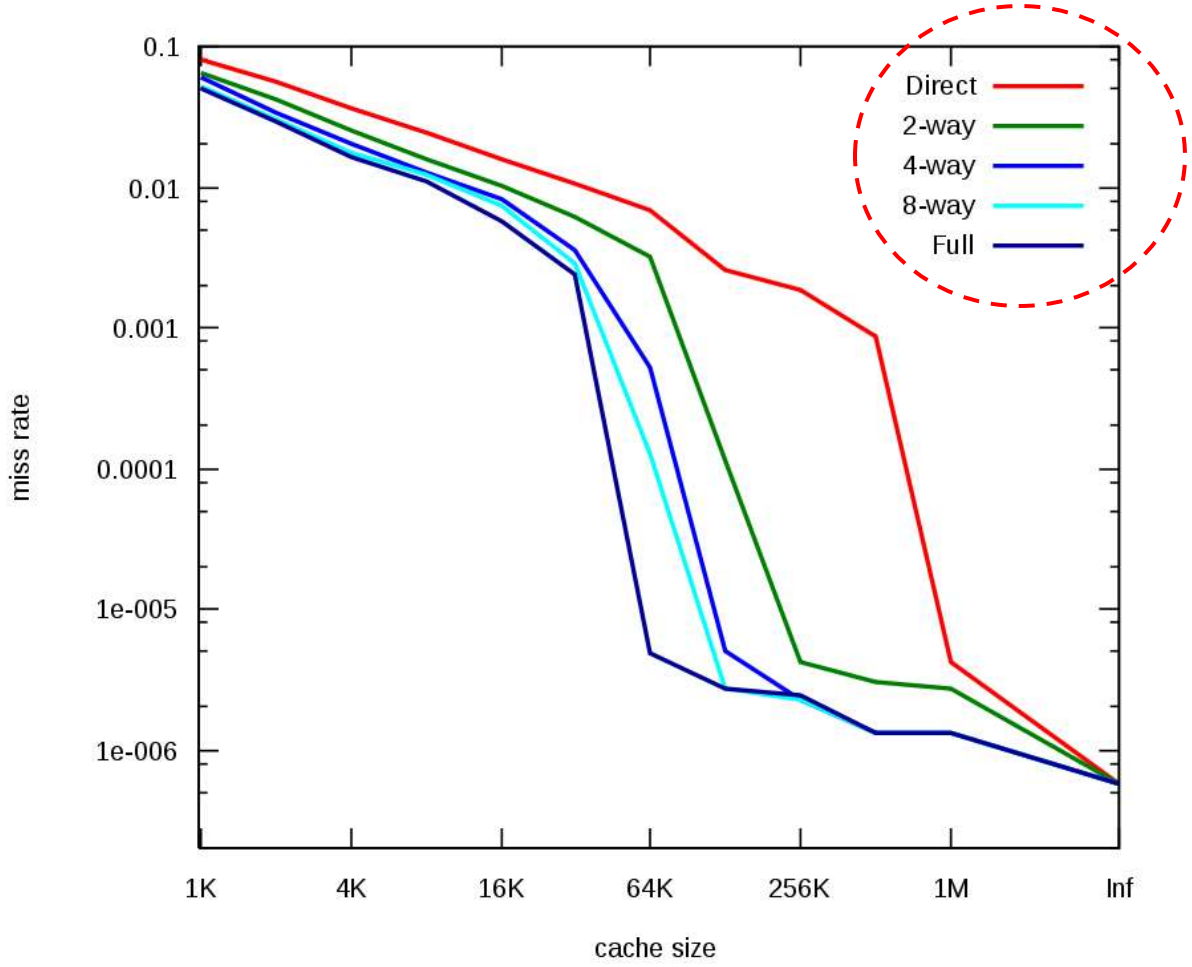
**V povprečju** pospeši dostop do delovnega pomnilnika.



# Podatki v predpomnilniku

- Podatek v predpomnilniku (**cache line**) je običajno dolg (8-512 bytov)
- Vsak podatek je označen z oznako (**tag**) – oznaka je del naslova (najvišji biti) izvirnega podatka v delovnem pomnilniku
- Ko procesor dostopa do pomnilnika, krmilnik predpomnilnika preveri, če je zahtevani podatek na voljo v predpomnilniku
- Če je podatek na voljo v predpomnilniku (**cache hit**) procesor dobi podatek iz predpomnilnika (kar se zgodi zelo hitro - s hitrostjo jedra procesorja)
- Če ga ni (**cache miss**), krmilnik prebere podatek iz delovnega pomnilnika (hitrost delovnega pomnilnika + morebitne zakasnitev prenosa, npr. pri SDRAM)
- Ob branju podatka iz delovnega pomnilnika ga krmilnik zapiše tudi v predpomnilnik. Pri tem nek starejši podatek izpade iz predpomnilnika. Pravila za nadomeščanje starih podatkov - **replacement policy**.
- Ko procesor želi pisati v pomnilnik, krmilnik posodobi podatek v predpomnilniku in ga uvrsti v **čakalno vrsto za zapis v delovni pomnilnik**.
- Vsake toliko časa krmilnik prepiše podatke iz predpomnilnika v delovni pomnilnik. Pravila za zapis v delovni pomnilnik – **write policy**.

# Učinkovitost predpomnilnika




Načini povezovanja vnosov v predpomnilniku z naslovi v delovnem pomnilniku

**Cache performance of SPEC CPU2000**, http://www.cs.wisc.edu/multifacet/misc/spec2000cache-data/ (2010-05-02)

# Povprečen dostopni čas

- Dostopni čas do RAMa = $t_R$
- Dostopni čas do predpomnilnika = $t_C$
- Delež dostopov, ko podatka ni v predpomnilniku (miss rate) = M

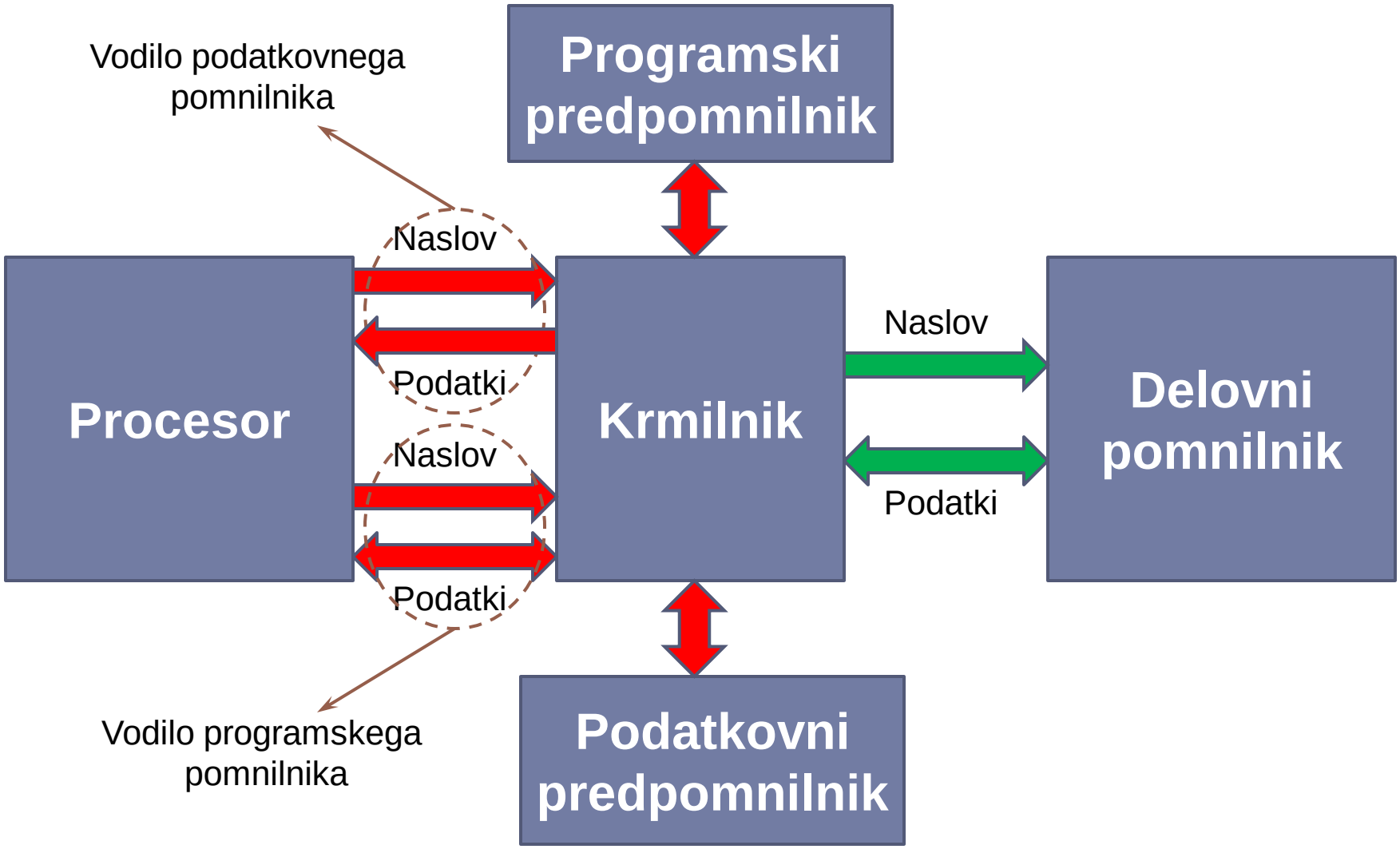
$$t = t_C + t_R \times M$$

- Primer:
  dostopni čas do RAMa: 80ns
  dostopni čas do predpomnilnika: 5ns
  miss rate: 0.01

  povprečen dostopni čas: 5ns + 80ns x 0.01 = 5.8ns

- Velika izboljšava v primerjavi s procesorjem, ki nima predpomnilnika (80ns dostopni čas).

# Spremenjena Harvardska arhitektura



# Spremenjena Harvardska arhitektura

- En sam naslovni prostor, dva predpomnilnika
- Na strani procesorja imamo Harvardsko arhitekturo
- Programer vidi samo en naslovni prostor
- Omogoča pisanje v “programski pomnilnik”, ki je potrebno za delovanje **just-in-time (JIT)** prevajalnikov (npr. Java Virtual Machine, Microsoft .NET)
- Omogoča rabo vsebine “programskega pomnilnika” v računskih operacijah (npr. konstante imamo lahko shranjene v programskem pomnilniku)
- Če spreminjamo program v pomnilniku, gre pisanje preko podatkovnega predpomnilnika, programski predpomnilnik pa ne ve, da se je program spremenil (vsebina je zastarela) **Cache Coherence Problem**

# Predpomnilnik v sodobnih mikroprocesorjih

- Ponavadi je predpomnilnik s krmilnikom že vgrajen v procesor
- Več nivojev predpomnilnika
- Primer:
  Intel Core i7-975, 3.33GHZ
  4 jedra (procesorji) v enem i7-975
  L1: 32kiB+32KiB ... prvi nivo (na vsako jedro, 32KiB podatki, 32KiB program)
  L2: 256KiB ... drugi nivo (na vsako jedro)
  L3: 8MiB ... predpomnilnik tretjega nivoja (skupen vsem jedrom)
- Kako je s tem pri LPC2138 (š-ARM)
  FLASH pomnilnik je počasnejši od jedra
  Primitiven predpomnilnik – MAM (Memory Acceleration Module)
  128 bitov (štirje ukazi naprej), 128 bitov (štirje ukazi nazaj)
  128 bitov (štirje 32-bitni podatki)
  = skupaj 3x128 bitov