VLIW i EPIC mašine

VLIW/EPIC Architectures

- Very Long Instruction Word (VLIW)
  - Procesor inicira više operacija po ciklusu – npr. Instrukcija ima 4 operacije

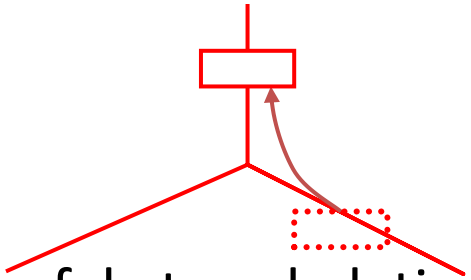
`r1 = L r4`	`r2 = Add r1,M`	`f1 = Mul f1,f2`	`r5 = Add r5,4`

  - Kompletno je specificirano u vreme prevođenja koje operacije ulaze u instrukciju
- Explicitly Parallel Instruction Computing (EPIC)
  - VLIW + Nove osobine
    - Predikacija (predikatsko izvršavanje, rotirajući registri, spekulativnost, ...

Podrška za spekulativnost po kontroli u vreme prevođenja

Spekulativnost po kontroli u vreme prevođenja – izvršavanje operacija koje u originalnom kôdu **možda** ne bi bile izvršavane. Možda, zbog ishoda grananja u vreme izvršavanja.

- Generalno se javlja zbog pomeranja kôda naviše preko uslovnih grananja.



- Transformacija je moguća ako se efekat spekulativne instrukcije može ignorisati ili se može uraditi nešto da se anulira njen efekat ako je tok izvršavanja bio kroz drugu granu (preimenovanje, kompenzacione operacije).
- **A izuzeci (exceptions)?**

Spekulativne operacije - izuzeci

Prevodilac obeležava spekulativne instrukcije, npr. sa E dodatim na naziv operacije.

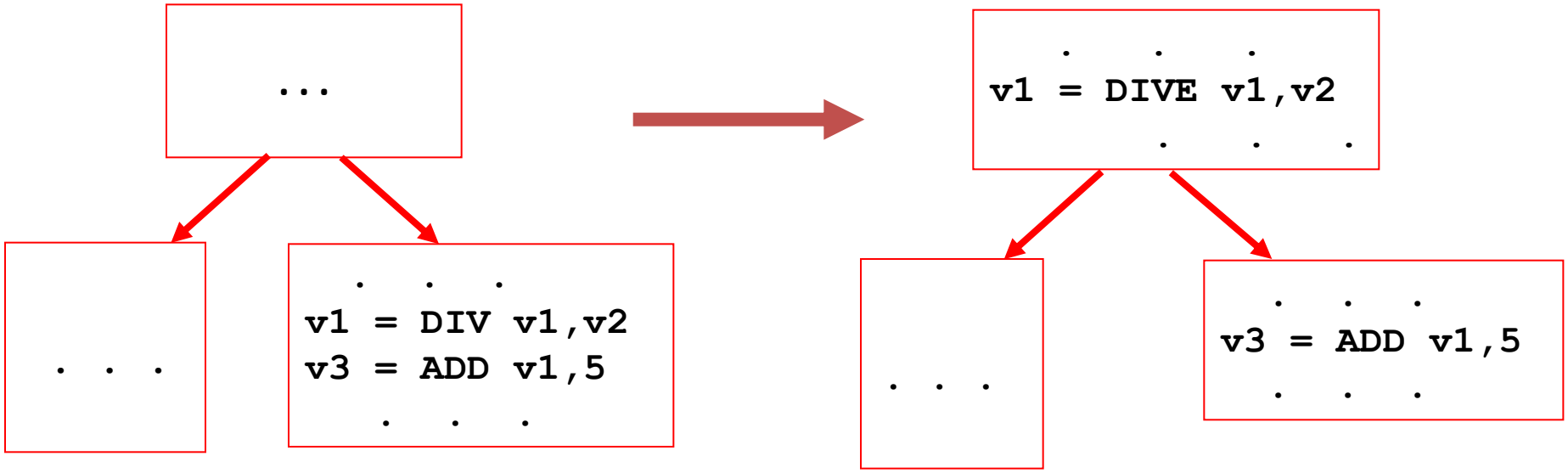
- npr. **`DIVE ADDE`**

Ako se desi izuzetak tokom spekulativne operacije, ne podiže se izuzetak, već se samo setuje dodatni bit za spekulativni izuzetak u registru rezultata, da zabeleži taj spekulativni izuzetak.

- Bit za spekulativni izuzetak se jednostavno propagira od strane spekulativnih instrukcija (argument sa setovanim bitom => rezultat)
- Kada instrukcija koja nije spekulativna pristupi registru sa postavljenim (setovanim) bitom, tek tada se podiže izuzetak.

Spekulativna Operacija primer

Optimizacija zahteva selidbu operacije preko grananja:



Ako se desi deljenje sa 0, izuzetak se podiže na mestu operacije ADD, gde je v1 operand (propagacija exception bita)

Preciznost izuzetka

- Mesto generisanja izuzetka je bazični blok iz koga je izvučena spekulativna operacija ili bazični blok posle tog bazičnog bloka, ali veoma blizu, na dinamičkom tragu.
- Izuzetak nije precizan, na nivou operacije, ali je uvek generisan, ako je bilo neophodno da se generiše, a nikada, kada nije potrebno da se generiše.
- Nepreciznost otežava obradu prekida

Predikacija sa jednim bitom

- Za većinu operacija se može uraditi predikacija
  - Mogu da imaju dodatni operand koji je jednobitni predikatski registar.

```
r2 = ADD r1,r3  if p2
```

  - Ako predikatski registar sadrži 0, operacija se ne izvršava
  - Vrednosti predikatskih registara se tipično postavljaju sa “compare-to-predicate” operacijama

```
p1 = CMPP<= r4,r5
```

Upotreba Predikacije

- Predikacija – najjednostvavnija forma:
  - if-konverzija
- Podrška pomeranju kôda – selidbama operacija preko IF.
  - npr. Hiperblokovi (hyperblocks)
- Sa kompleksnijim compare-to-predicate operacijama, dobija se redukcija visine kontrolnih zavisnosti

If-konverzija

- If-konverzija zamenjuje uslovne skokove sa operacijama sa predikatom, a primer je kôd za:

```
if (a < b)
  c = a;
else
  c = b;
if (d < e)
  f = d;
else
  f = e;
```

se može predstaviti sa dve EPIC instrukcije:

`P1 = CMPP.< a,b`	`P2 = CMPP.>= a,b`	`P3 = CMPP.< d,e`	`P4 = CMPP.>= d,e`
`c = a if p1`	`c = b if p2`	`f = d if p3`	`f = e if p4`

Compare-to-predicate instrukcije

- Na prethodnom slajdu su bila dva para gotovo identičnih instrukcija koje samo računaju komplemente
  - Obezbeđuju se simultano dva izlaza CMPP instrukcije
  - p1,p2 = CMPP.W.<.UN.UC r1,r2
  - U označava bezuslovno, N znači normalno, a C označava komplement
  - Postoje i druge opcije (npr. or, and)

If-konverzija, ponovo

- Sada samo dve dvoizlazne CMPP instrukcije i generisani kôd za:

```
if (a < b)
  c = a;
else
  c = b;
if (d < e)
  f = d;
else
  f = e;
```

može da bude:

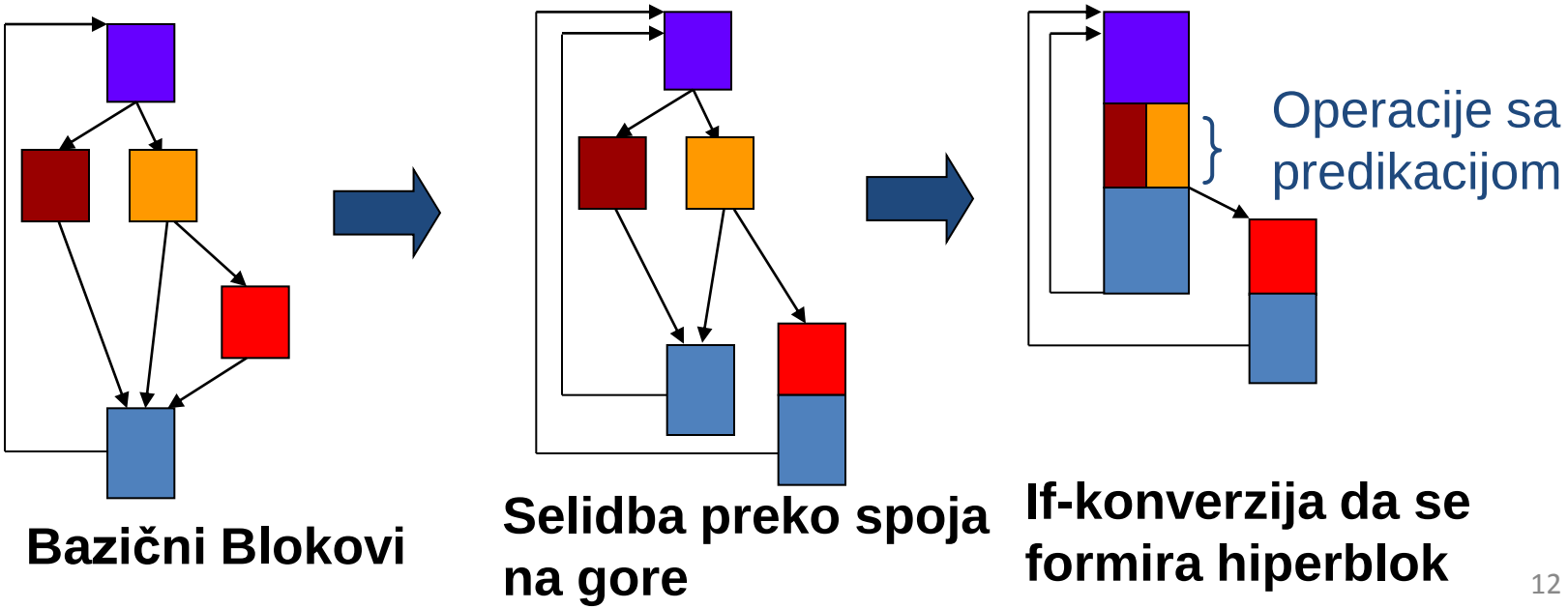
Samo dve CMPP instrukcije su sada neophodne

```
p1,p2  = CMPP.W.<.UN.UC a,b | p3,p4  = CMPP.W.<.UN.UC d,e
```

```
c = a   if p1 | c = b   if p2 | f = d   if p3 | f = e   if p4
```

Formiranje Hiperblokova

- If-konverzija se koristi za pravljenje većih blokova operacija od običnih bazičnih blokova
  - Radi se selidba preko spoja na gore za cele bazične blokove (tail duplication), da bi se uklonili spojevi u sredini hiperbloka
  - if-konverzija se sprovodi nakon toga
  - Veći blokovi pružaju veći paralelizam!



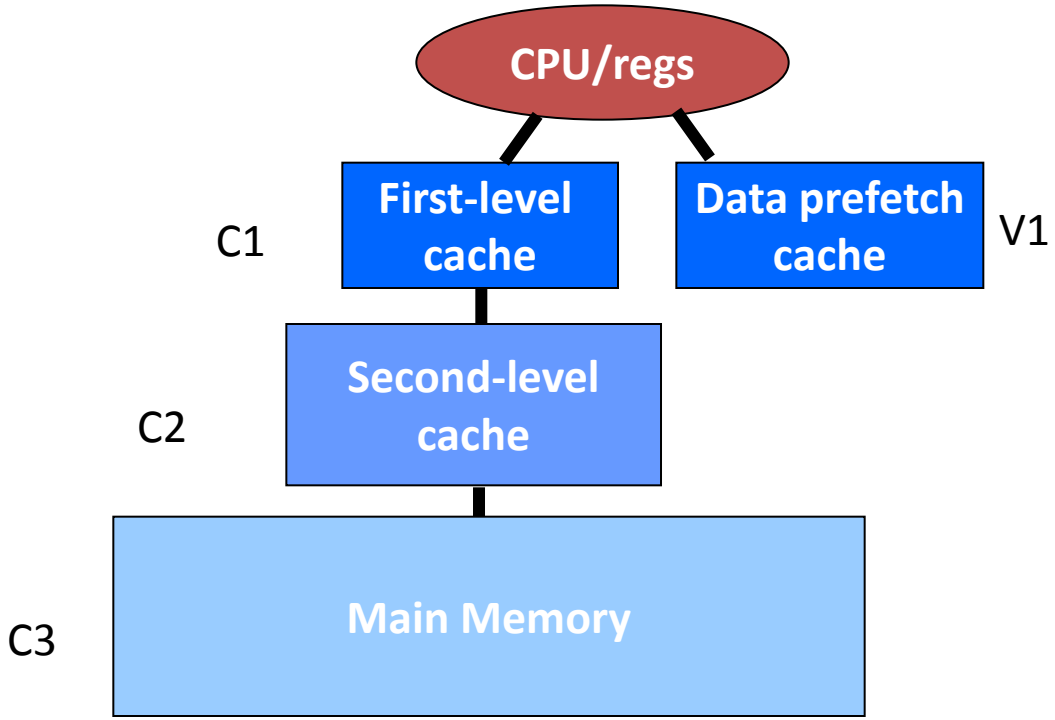
EPIC Memorijska Hijerarhija

Može da bude vidljiva za kompajler.

- Kod store instrukcija, kompajler može da specificira u koji cache podaci treba da se smeste.
- Kod load instrukcija, kompajler može da specificira u kom cache-u se podaci nalaze i u koji cache podaci treba da budu smešteni.

Ovim je podržano statičko raspoređivanje load/store operacija sa očekivanjem da su pretpostavljena kašnjenja tada tačno određena.

Memorijska Hijerarhija

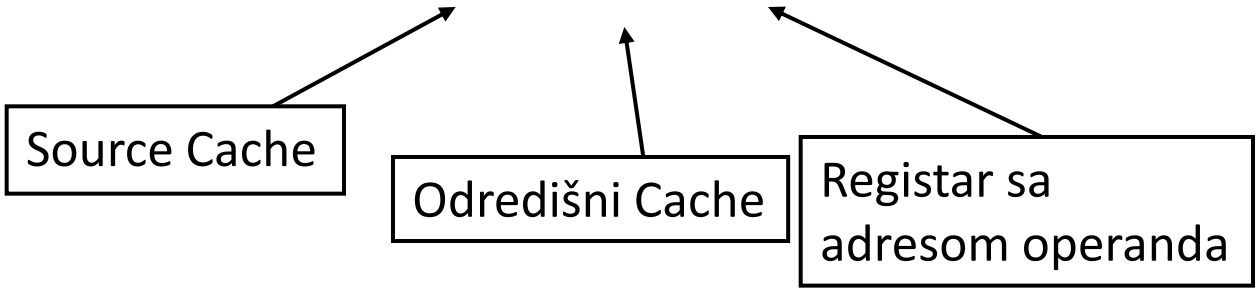


data-prefetch cache

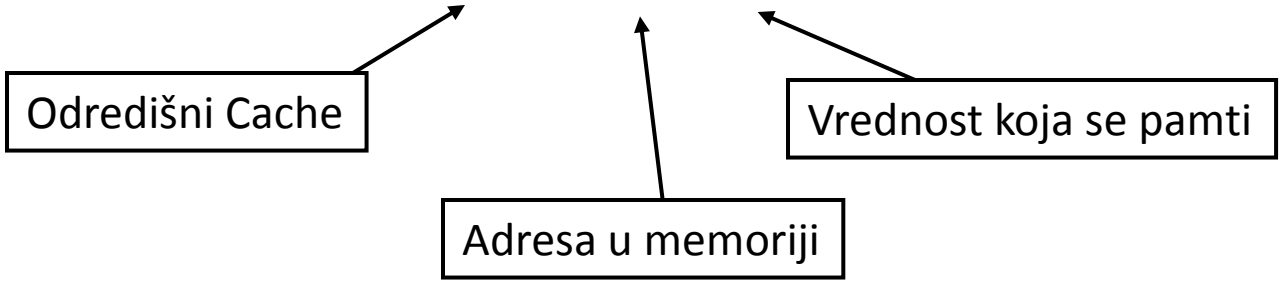
- Nezavistan od cache prvog nivoa
- Koristi se za pamćenje veće količine podataka koja može da spreči prevelike interakcije u memorijskoj hijerarhiji
- Ne zahteva mehanizme zamene u cache-u

Load/Store Instrukcije

Load Instruction: `r1 = L.W.C2.V1 r2`



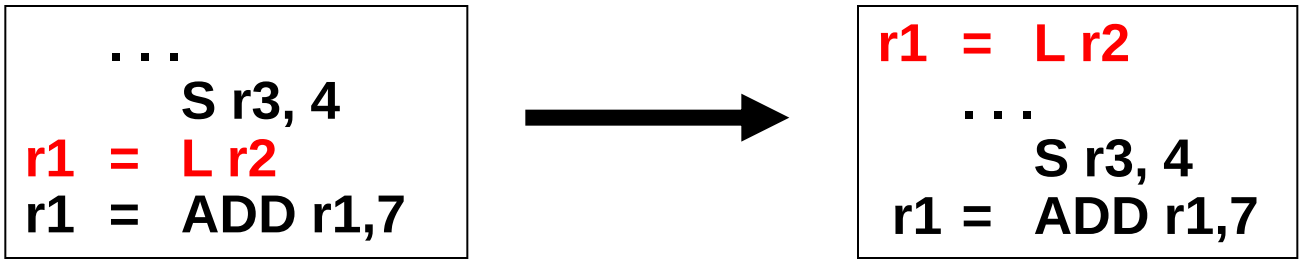
Store Instruction: `S.W.C1 r2,r3`



Izvorišni cache se mora korektno definisati.

Run-time Memory Disambiguation

Zbog velikog kašnjenja kod load-a, ovo je željena optimizacija:



Međutim, ovo nije validno ako load i store referenciraju istu lokaciju, tj. r2 i r3 sadrže istu adresu. Ovo važi kada se u vreme prevođenja ne može odrediti da li su iste adrese.

Može se obezbediti *run-time memory disambiguation*, odnosno određivanje da li postoji zavisnost u vreme izvršavanja.

Run-time Memory Disambiguation (2)

Dve specijalne instrukcije u zamenu za jednu load instruction:

```
r1 = LDS r2        ; spekulativni load
```

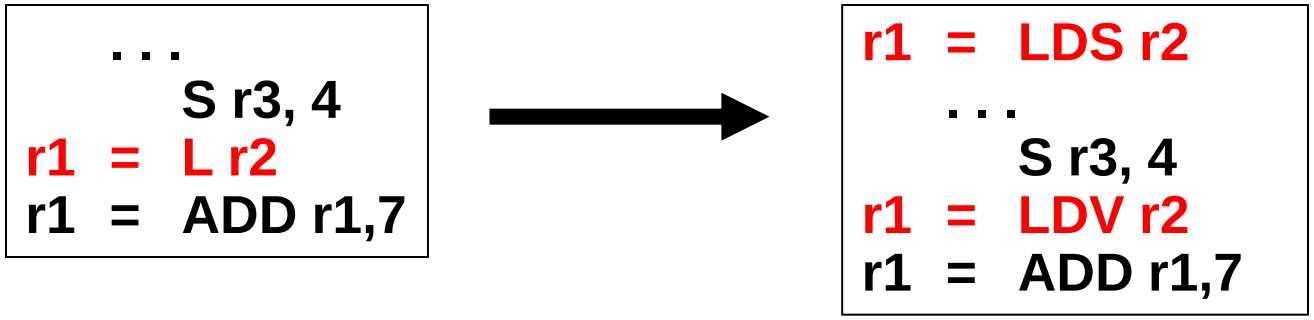
Započinje load kao normalnu operaciju i pamti vrednost u posebno mesto (tabelu) za pamćenje vrednosti iz memorijske lokacije

```
r1 = LDV r2    ; load verify
```

- Proverava da li je možda bilo pamćenja u tu lokaciju od kada je LDS započeta.
- Ako se to dogodilo, koči se pipeline i novi load se izdaje. U suprotnom je ekvivalentno sa no-op (pogodili smo u spekulaciji).

Run-time Memory Disambiguation (cont)

Finalno, prethodna optimizacija postaje



Zaključci

- Moderne VLIW/EPIC arhitekture pružaju široku paletu mogućnosti i zahtevaju veoma sofisticirane optimizacije
- Predikacija je moćna osobina ovih mašina
- Dinamičko memorijsko adresiranje ne mora da sprečava optimizaciju (paralelizaciju) zahvaljujući spekulativnom load-u