Paralelizam - instrukcijski nivo

Instrukcijski nivo paralelizma

- Specificirani redosled operacija - korektno izvršavanje programa.
- **Paralelizam može da promeni redosled operacija.**
- Određivanje redosleda izvršenja operacija koji je različit u odnosu na redosled izvršenja **specificiran od strane programera.**
- Redosled mora biti takav da generisani rezultati u oba slučaja (sekvencijalni i paralelni) moraju biti uvek isti.

Graf i zavisnosti

- Kada dogadjaj A mora da se desi pre dogadjaja B, kažemo da B zavisi od A. Odnos izmedju ova dva dogadjaja naziva se **zavisnost (dependency).**
- Ponekad zavisnosti mogu da postoje zbog izračunavanja ili memorijskih operacija – zavisnosti po podacima (data dependencies).
- U slučajevima kada se čeka na grananje (branch) ili se čeka da se izadje iz izvršenja petlje – upravljačka zavisnost (control dependency).
- Cilj je da se što više eliminišu zavisnosti. Preuređenje programa treba izvesti tako da se što više operacija izvrši u paraleli (koliko to dozvoljavaju resursi mašine) u cilju skraćenja ukupnog vremena izvršavanja,a da se ostatak operacija čije izvršavanje je ograničeno odgovarajućim zavisnostima, održe u pravom redosledu.
- Jedan od načina za efikasniju analizu sekvence instrukcija je organizacija te sekvence u formi usmerenog acikličnog grafa (directed acyclic graph - DAG). Na sličan način kao i instrukcije tako i DAG opisuje sva izračunavanja i ukazuje na sve odnose koji postoje izmedju instrukcija.

Prave zavisnosti po podacima

- Prva operacija smešta podatak u lokaciju koja se kasnije čita od strane drugoe operacije:

  S1 X=.....

  S2 .....=X

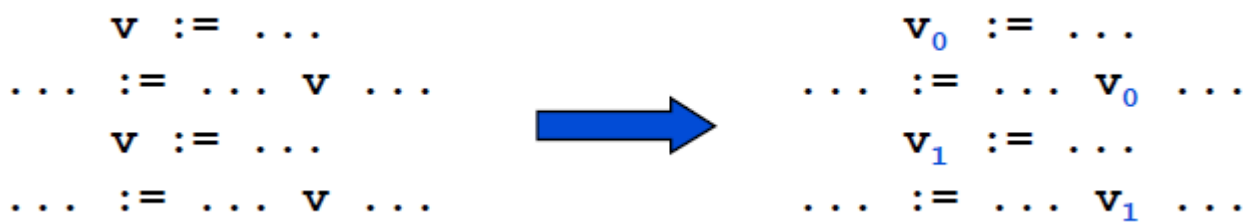
- Bez obzira na raspored mora se obezbediti da druga instrukcija dobije vrednost izračunatu od strane prve.
- Mogu biti direktne ili indirektne

Antizavisnosti po podacima

- Prva operacija čita sa lokacije u koju druga operacija kasnije smešta vrednost
- S1 .....=X
- S2 X=.....
- Nova vrednost ne sme biti upisana u lokaciju pre nego što poslednja predviđena instrukcija pročita podatak sa te lokacije
- Može se eliminisati preimenovanjem

SSA forma (1)

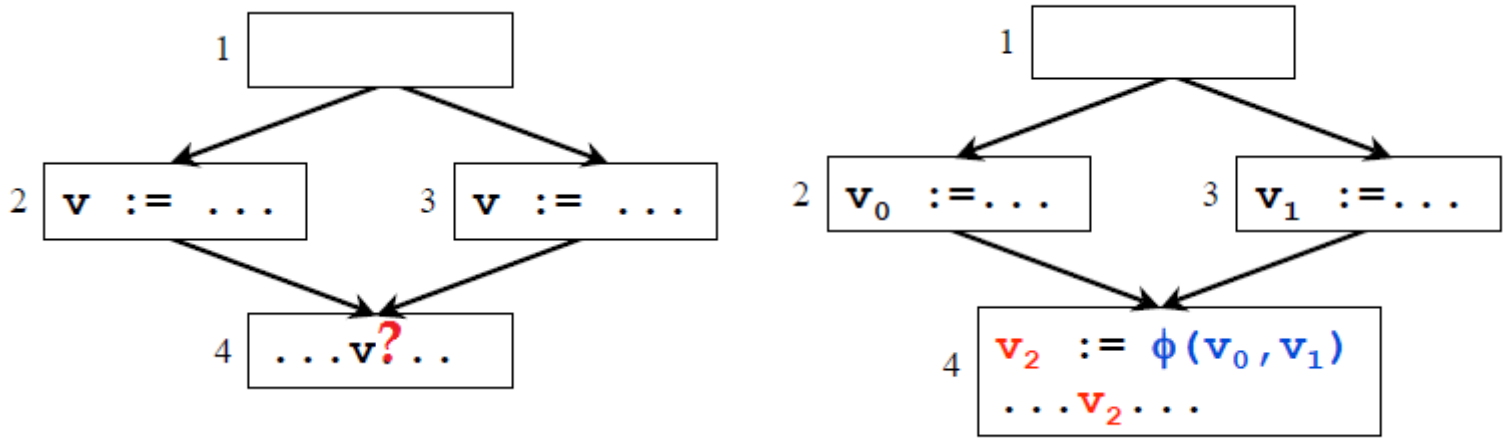
- Svakoj promenljivoj vrednost može biti dodeljena samo jednom (jedna definicija)
- Funkcionalno programiranje
- Memorija alocirana za tu promenljivu može se dealocirati nakon poslednjeg čitanja
- U neizmenjenom poretku instrukcija preimenovanje se radi pri svakom upisu i za sve instrukcije koje čitaju tu vrednost



- Jednostavno na nivou bazičnog bloka, ali kompleksnije na nivou dijagrama

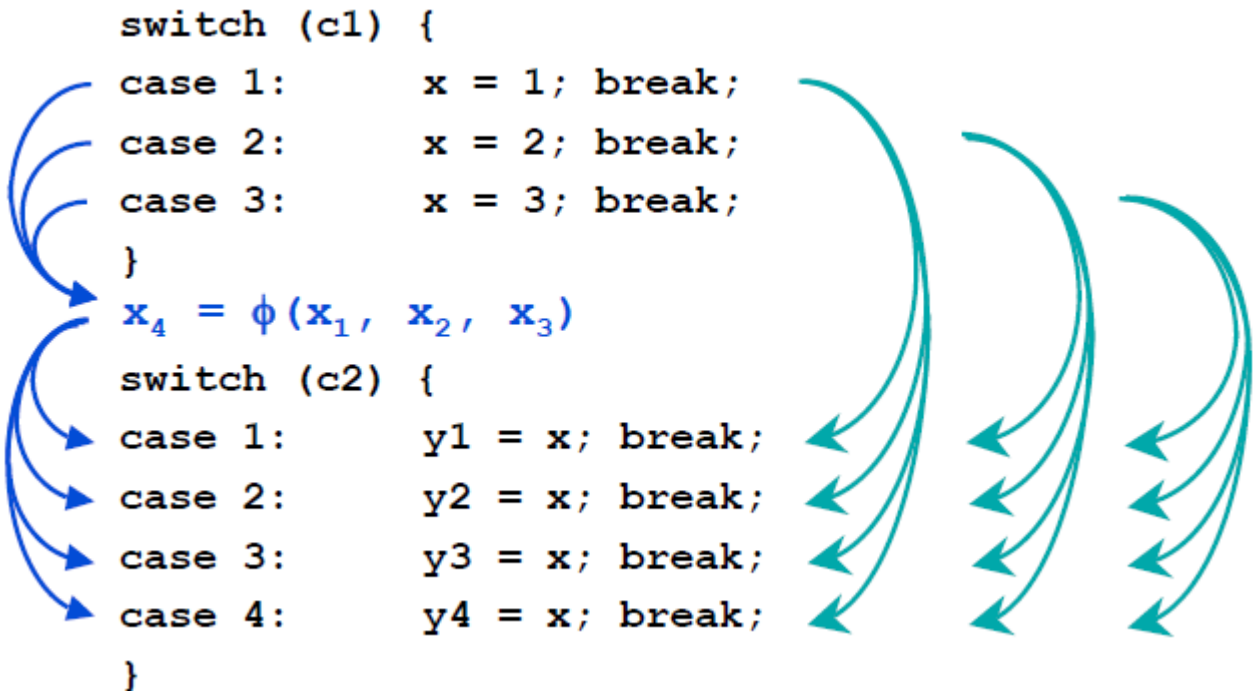
SSA forma (2)

- Vrednost neke promenljive se može dobiti na više načina i zavisi od izvršavanja programa



- Φ funkcije određuju koja vrednost će se naći u lokaciji

SSA forma (3)


switch (c1) {
case 1: x = 1; break;
case 2: x = 2; break;
case 3: x = 3; break;
}
$x_4 = \phi(x_1, x_2, x_3)$
switch (c2) {
case 1: y1 = x; break;
case 2: y2 = x; break;
case 3: y3 = x; break;
case 4: y4 = x; break;
}

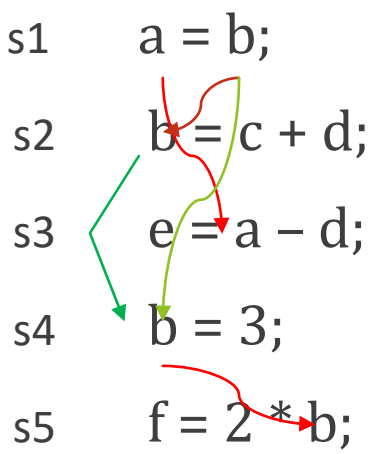
Izlazne zavisnosti po podacima

- Obe operacije upisuju u istu lokaciju
- S1 X=...

  ...

  S2 X=...
- Izmena redosleda instrukcija rezultovala bi pogrešnom vrednošću smeštenom u lokaciji
- Na nivou bazičnog bloka ako ne postoje instrukcije između S1 i S2, S1 se može eliminisati kao mrtav kod, a u suprotnom izlazna zavisnost predstavlja tranzitivnu zavisnost za par prava zavisnost - antizavisnost

Zavisnosti po podacima



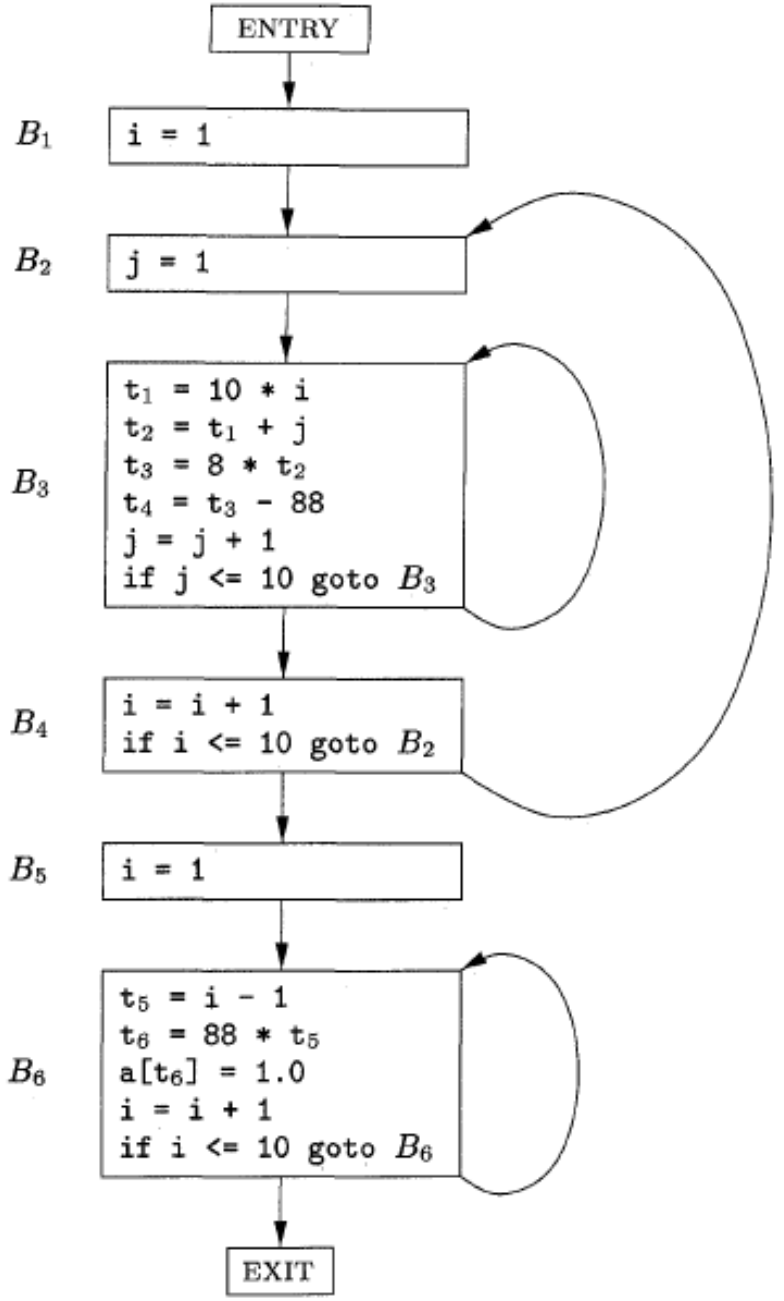
prave zavisnosti

antizavisnosti

izlazne zavisnosti

Bazični blok

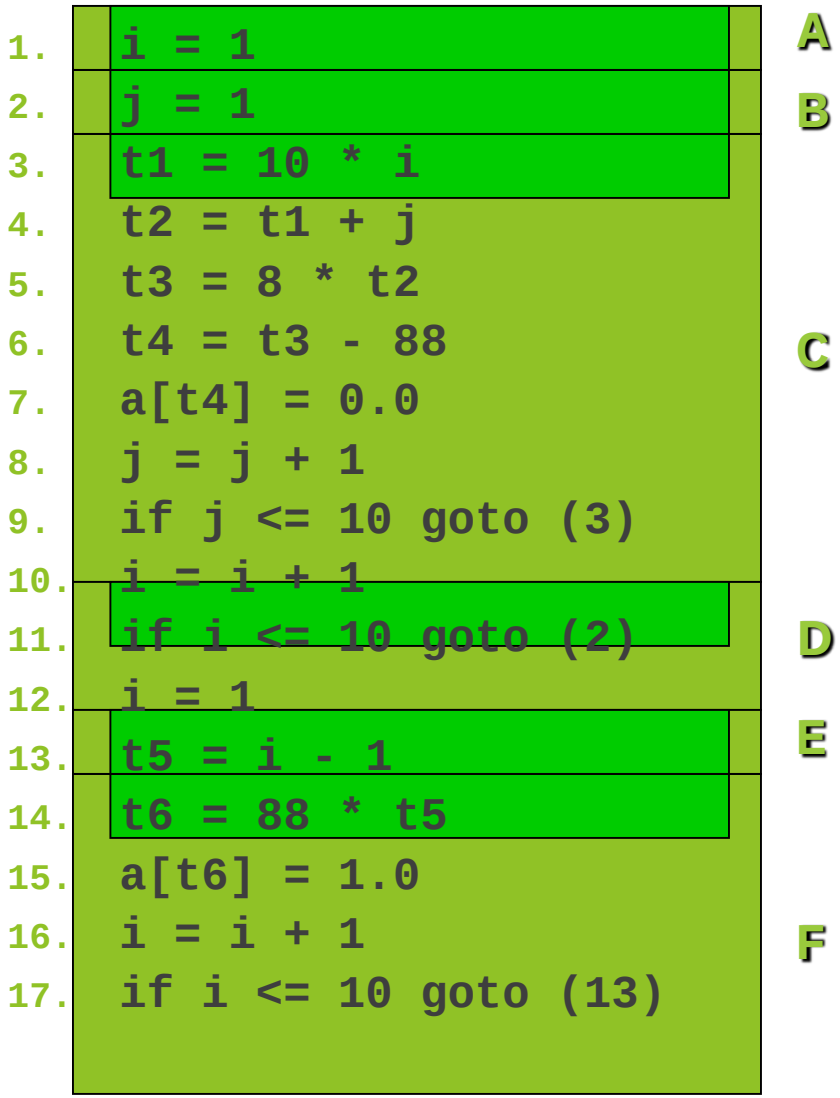
- Sekvenca od 1 ili više uzastopnih instrukcija
- Prva instrukcija je ili prva instrukcija programa ili odredište skoka u programu (grananje, poziv funkcije, ...) ili se nalazi u programu kao prva instrukcija nakon skoka.
- Poslednja instrukcija je ili instrukcija iskakanja iz bazičnog bloka ili poslednja instrukcija u programu.
- Bazični blokovi predstavljaju čvorove **control flow** dijagrama. Grane koje povezuju čvorove su moguće putanje u *runtime*-u.



Bazični blok

```
leaders = {1}    // start of program
for i = 1 to |n|    // all instructions
   if instr(i) is a branch
    leaders = leaders U  targets of instr(i) U
   instr(i+1)
worklist = leaders
While worklist not empty
   x = first instruction in worklist
   worklist = worklist – {x}
   block(x) = {x}
   for i = x + 1; i <= |n| && i not in leaders; i++
    block(x) = block(x) U {i}
```

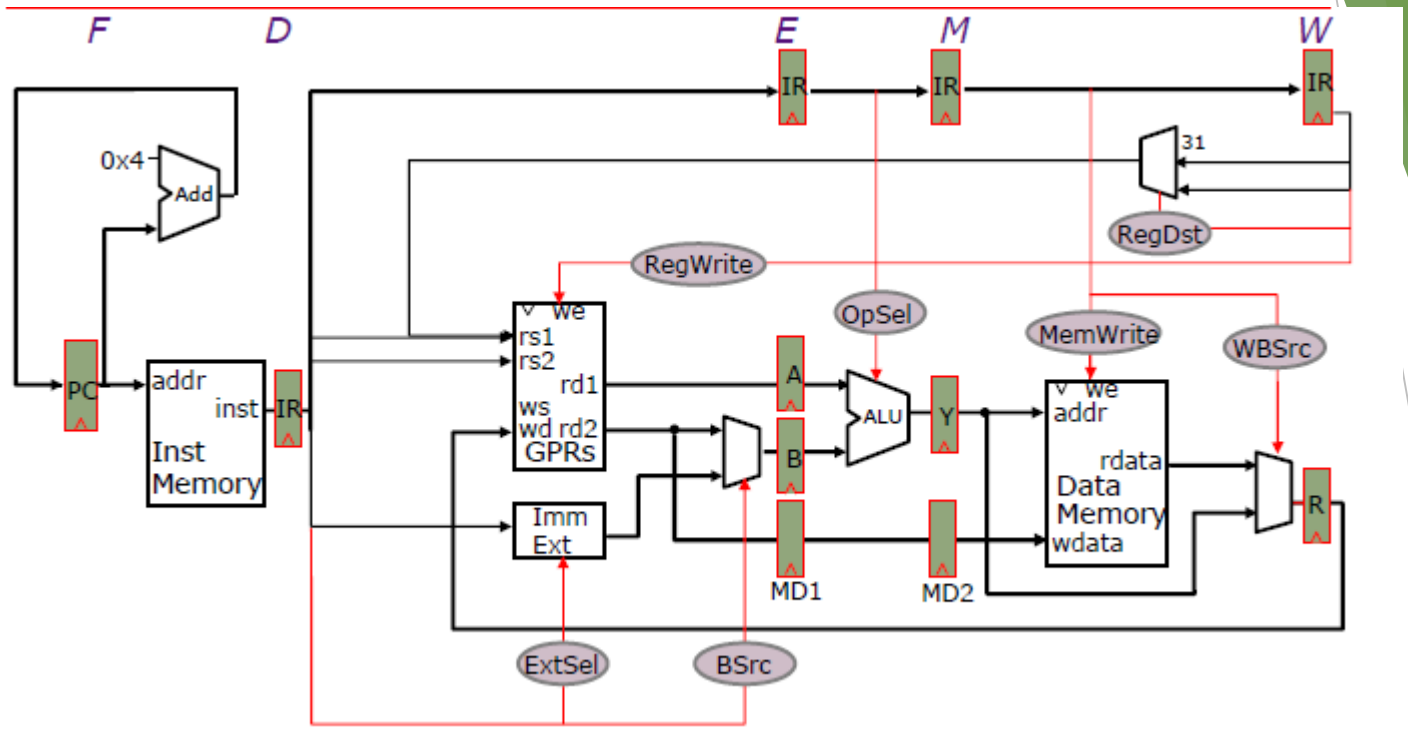
Bazični blokovi primer



Vodeće instrukcije

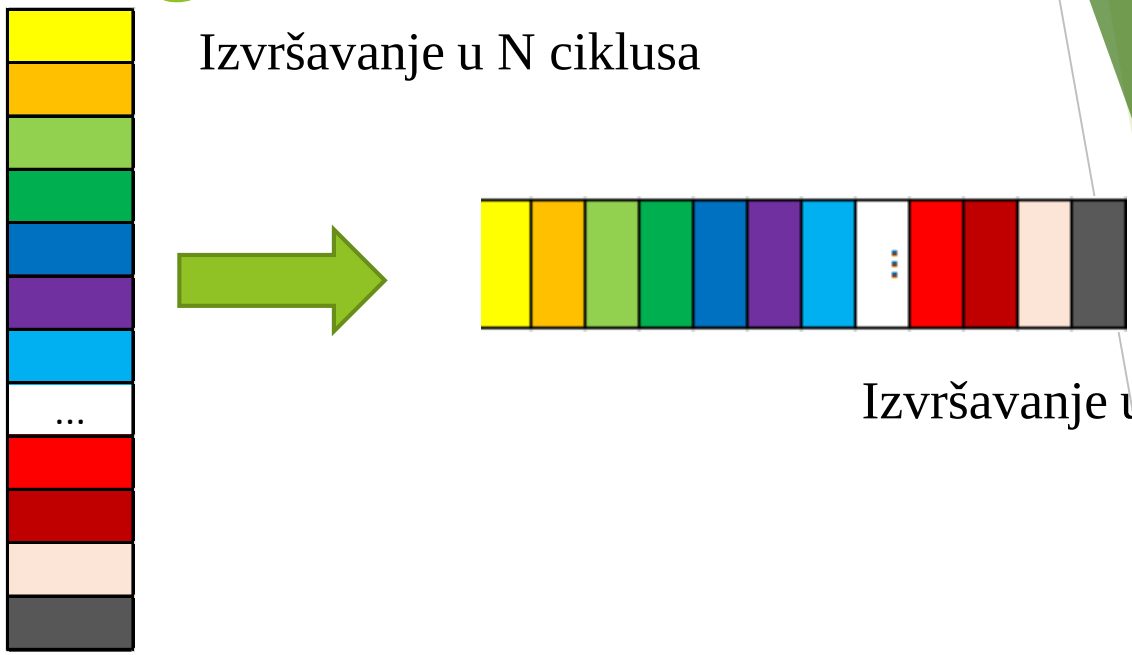
Bazični blokovi

Pipeline



- **U kontekstu projektovanja hardvera, zavisnosti se često nazivaju hazardi (hazards) prvenstveno zbog njihovog efekta na protočni rad sistema.**
- U tom smislu prava zavisnost je identična kao i RAW (read after write) hazard, antizavisnost je ekvivalentna sa WAR (write after read) hazardom, dok se izlazna zavisnost naziva WAW (write after write) hazard

Formiranje optimalnog koda



Ograničavajući hardverski resursi, zavisnosti između instrukcija, ...

- Optimizacija koda na više nivoa
- Opimizacija na nivou bazičnog bloka – List Scheduling algoritam

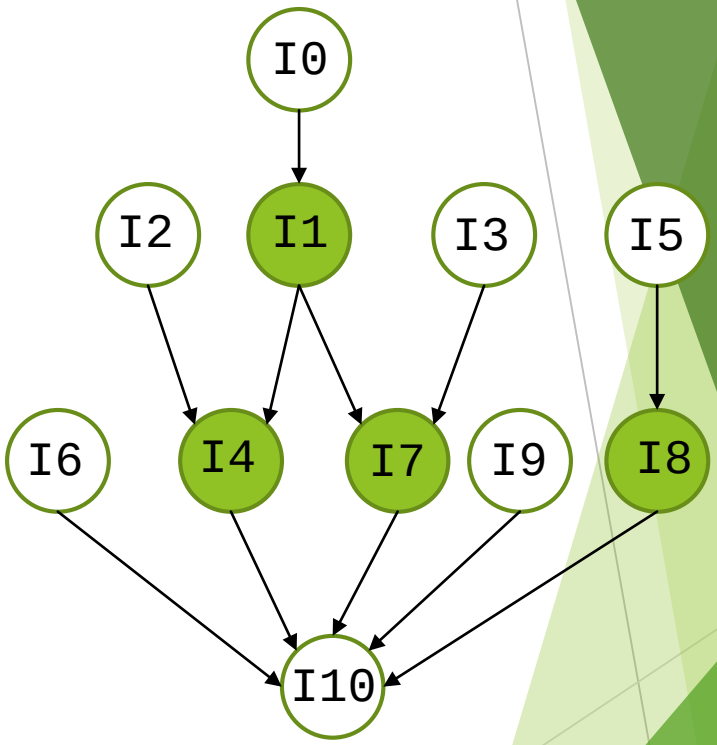
List Scheduling algoritam (1)

- Lista operacija koje mogu da se izvrše (data ready operacije)
- Održavaju se skupovi raspoređenih i neraspoređenih operacija
- U skladu sa određenim kriterijumom, bira se jedan operacija iz liste, locira se najraniji ciklus njenog početka i proverava se da li u svim ciklusima postoje raspoloživi resursi za izvršavanje operacije
- Ukoliko su svi uslovi zadovoljeni, skupovi i lista se ažuriraju, a u suprotnom se izvršavanje odlaže za jedan ciklus i postupak provere ponavlja
- Različiti kriterijumi za izbor instrukcije iz data ready skupa mogu rezultovati različitom ukupnom dužinom trajanja izvršavanja koda

List Scheduling algoritam (2)

- U primeru operacija sabiranje traje dva ciklusa, a sve ostale operacije po jedan ciklus

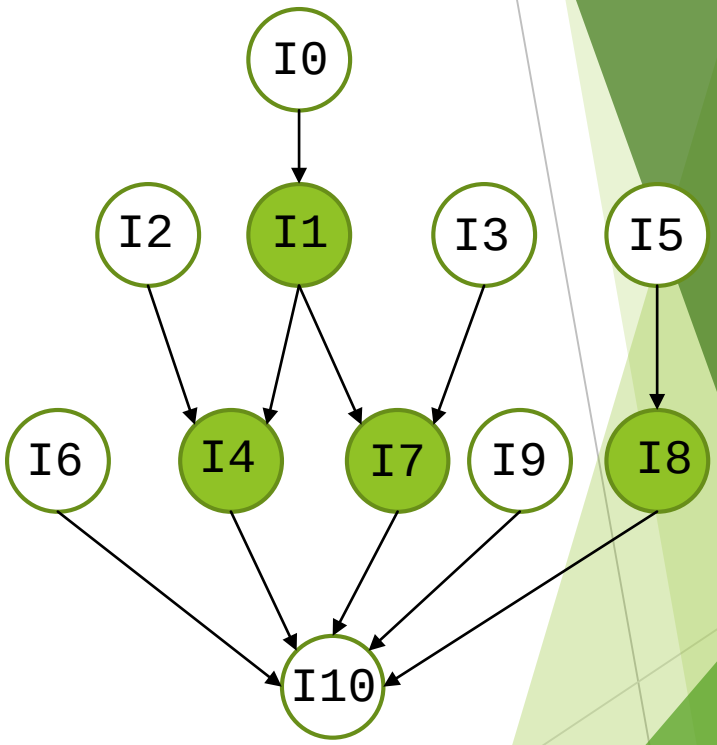
```
I0:  a = 1
I1:  f = a + x
I2:  b = 3
I3:  c = 10
I4:  g = f + b
I5:  d = 25
I6:  e = 18
I7:  h = f + c
I8:  j = d + y
I9:  z = 2
I10: jmp  L
```



List Scheduling algoritam (3)

- Najčešće izbor prema operacijama koje se nalaze na kritičnom putu
- Resursi se pune od vrha formiranog grafa

GR1	GR2	Ciklus
I0	I2	0
I1	I3	1
I5	I6	2
I4	I7	3
I8	I9	4
-	-	5
I10		6



List Scheduling algoritam (3)

- Primarno određivanje završnog trenutka operacije, a početak se koristi za određivanje raspoloživosti resursa.
- Posmatranje izvršavanja od poslednje operacije – obrnuti redosled
- Drugačiji rasporedi za različito vreme trajanja operacija (store u memoriju)
- Prave zavisnosti ograničavajuće za raspoređivanje na nivou bazičnog bloka

GR1	GR2	Ciklus
I0	I2	0
I1	I5	1
I3	I8	2
I4	I7	3
I6	I9	4
I10	-	5