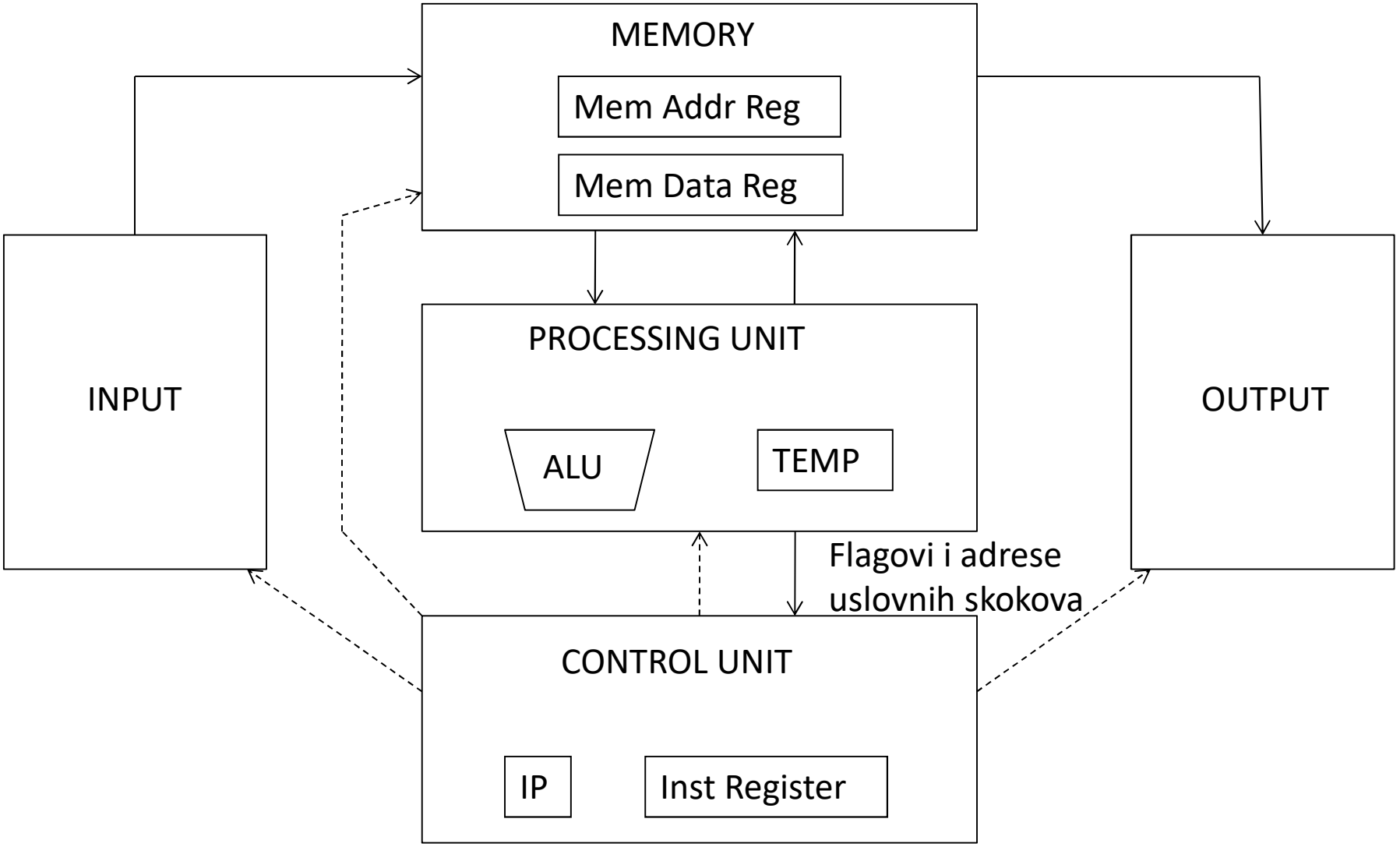
Data Flow mašine

Kontrolom pokretane mašine

- Konvencionalni modeli programa su kontrolom pokretani, jer kontrolna jedinica određuje koja se instrukcija naredna izvršava.
  - Sekvenca instrukcija je precizno određena
  - Kontrola određuje narednu instrukciju pomoću programskog brojača/registra (instruction pointer)
  - Ako nema skoka, naredna instrukcija po adresi u memoriji se sledeća izvršava.
  - Ako ima skoka, programom je definisano kako se određuje adresa naredne instrukcije (adresa skoka)

Von Neumann Model



Redosled izvršavanja za kontrolom pokretane mašine

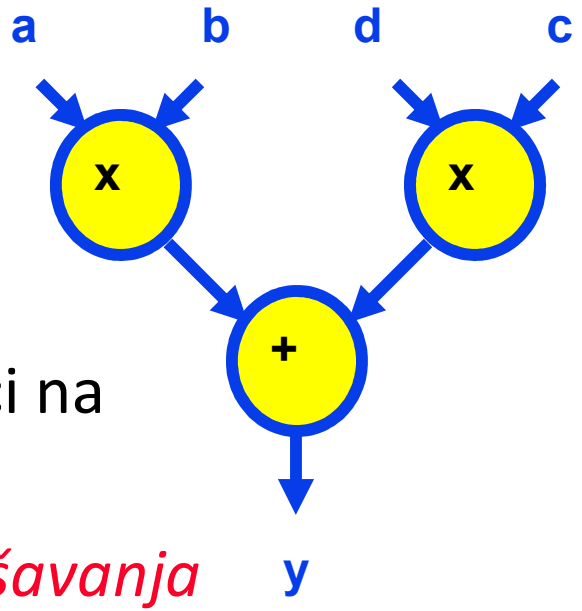
Izvrši instrukciju kada je prethodna po memoriji izvršena:

```
s1:     r = a*b;
s2:     s = c*d;
s3:     y = r + s
S4:     if s  y then SX ...
SY:     else ...;
```

...
s3 se izvršava kada je s2 završena
s4 se izvršava kada je s3 završena
SX ili SY se izvršava kada se završi S4
Ili SX ili SY je sledeća instrukcija u memoriji

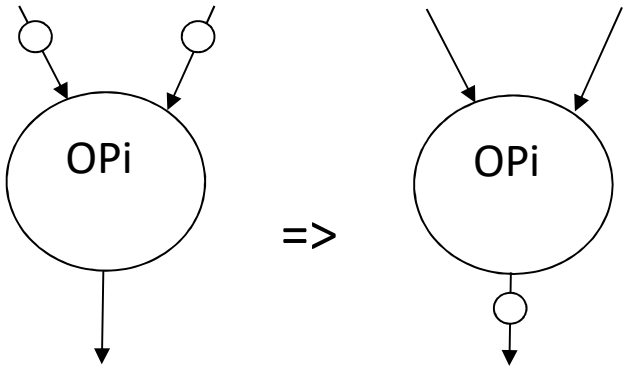
Data Flow - bazični blok

- Npr. izračunavanje y = a*b + c*d
- Predstavimo acikličkim grafom zavisnosti po podacima
  - Podaci putuju duž grana
- Pravilo izvršavanja:
  - Izvrši instrukciju kada su svi potrebni ulazni podaci na raspolaganju
  - *Pravilo o pokretanju izvršavanja po podacima*



Predstavljanje Data Flow programa i njegovog izvršavanja

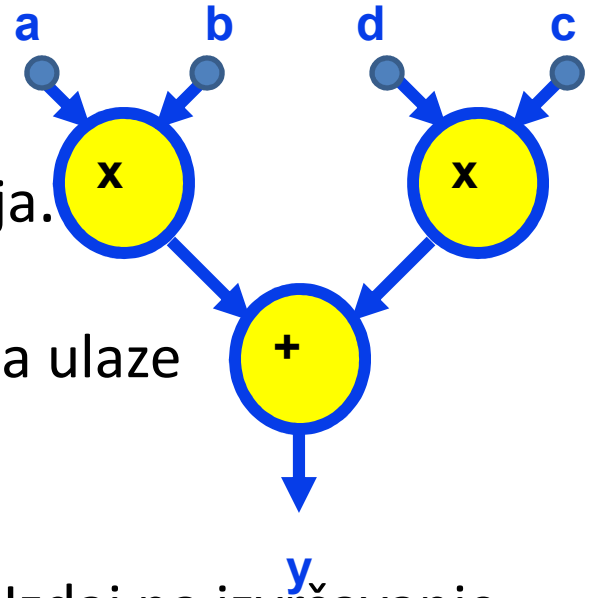
Prvo za bazični blok. Uvodi se operator primitivne operacije kojim se obezbeđuje izvršavanje



Kružići predstavljaju torke podataka (data tokeni) koje se pojavljuju kao inicijalne data ready vrednosti ulaza čvorova (primitivnih operacija) slobodnih na vrhu ili tokom izvršavanja kao izračunate vrednosti tokom izvršavanja programa

Data Flow

- Čvorove grafa zavisnosti po podacima zamenjujemo sa primitivnim operacijama, pa data flow firing rule automatski važi - instruction se **može izvršavati kada su ulazni data tokeni raspoloživi**
- Izvršavanjem primitivnih operacija se tokeni pojavljuju na izlazu i uklanjaju sa ulaza primitivnih operacija.
  - Sabiranje mora da čeka da se oba množenja izvrše i pošalju tokene na ulaze



- *Analiza zavisnosti po podacima potrebna u vreme prevođenja.*
  - Jedinice za izvršavanje instrukcija: Izdaj na izvršavanje instrukciju kojoj su oba argumenta upisana

Distribucija rezultata

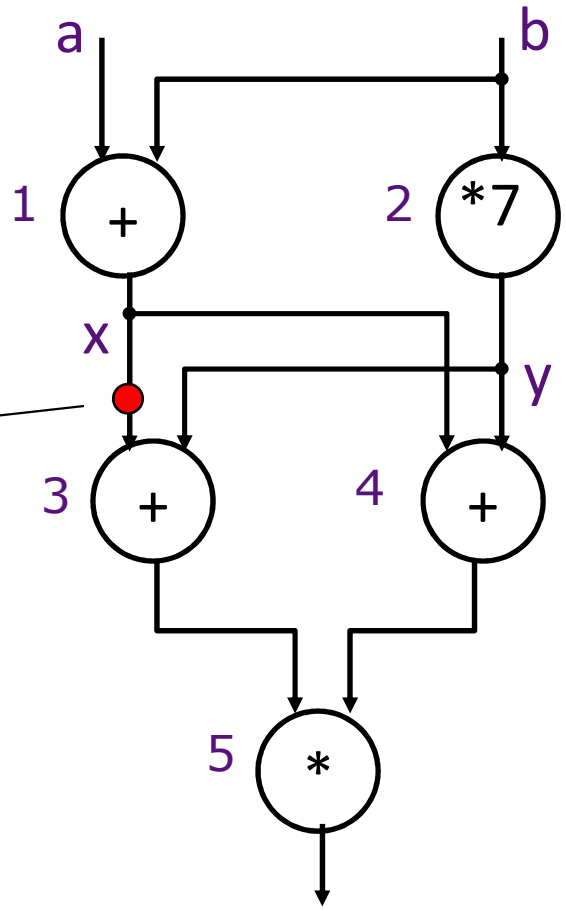
```
{x = a + b;
 y = b * 7;
 (x-y) * (x+y)}
```

- Vrednosti u grafu su predstavljene preko data tokena

data token < ip , p , v >

instruction ptr — port — data

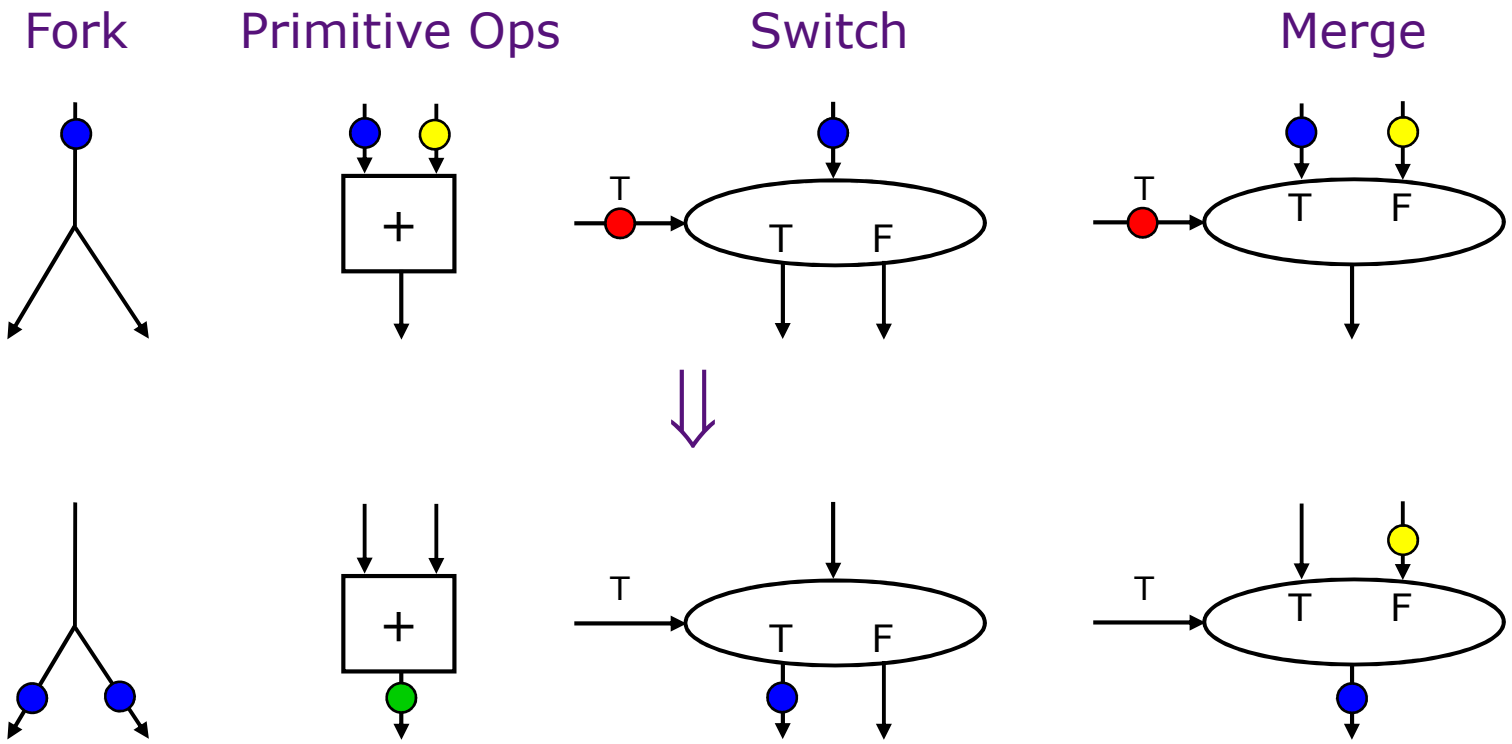
ip = 3
p = L



- Kopije rezultata moraju da se distribuiraju do odgovarajućih portova svih destinacionih operacija
- Instruction ptr je adresa tokena operacije (instrukcije) u memoriji

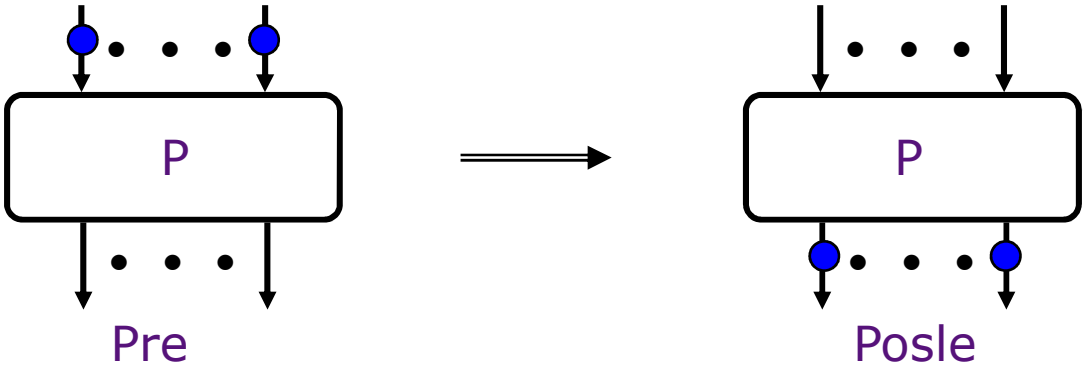
Dataflow Operatori

- Mali skup operatora definiše generalni programski jezik – istom bojom su kodirane iste vrednosti value dela data tokena



Dodatni operatori

Barijera

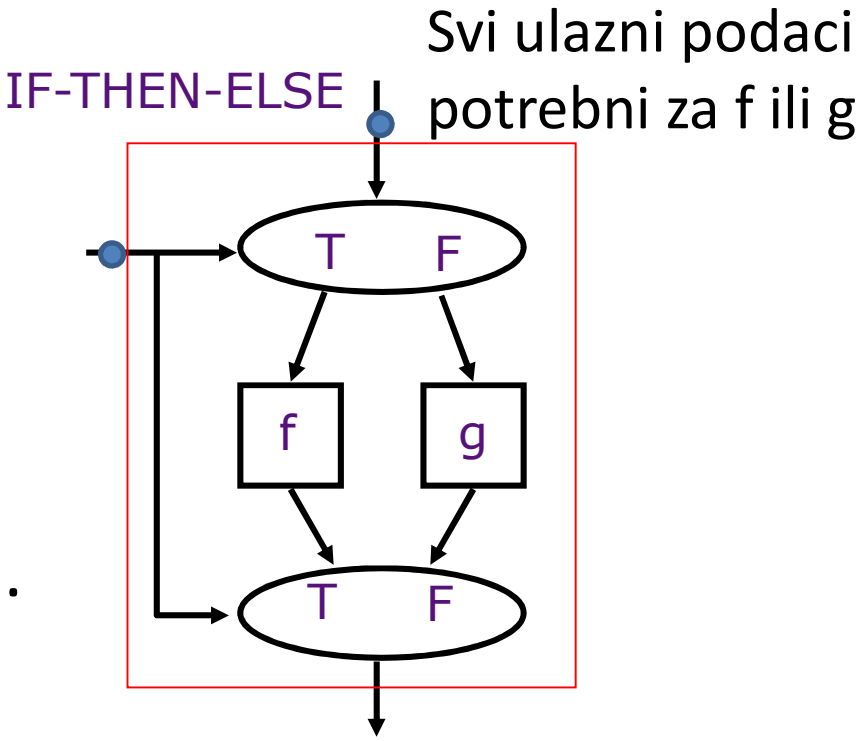


Kôd bez petlji

- Potrebno je uvesti kontrolne zavisnosti
- Kontrolne zavisnosti se moraju pretočiti u zavisnosti operatora ili primitivnih operacija od ulaznog podatka
- Za to služe switch i merge operatori
- Switch služi da se ulazni token podatka na data ulazu dovede do ulaza operacija samo jedne grane u kôdu, zavisno od vrednosti tokena na kontrolnom ulazu (flag uslovnog skoka).
- Merge bira rezultate iz iste grane. Za Merge je dovoljno za prosleđivanje tokena da budu raspoloživi kontrolni ulaz i samo izabrani ulaz

IF-THEN-ELSE

Flag upravlja distribucijom ulaznih argumenata ka delovima kôda f i g preko Switch operatora. Distribucija rezultata ili then ili else grane se obavlja preko merge operatora



Samo jedna grana se izvršava jer druga grana nema ulazne podatke!!!

Dataflow jezici

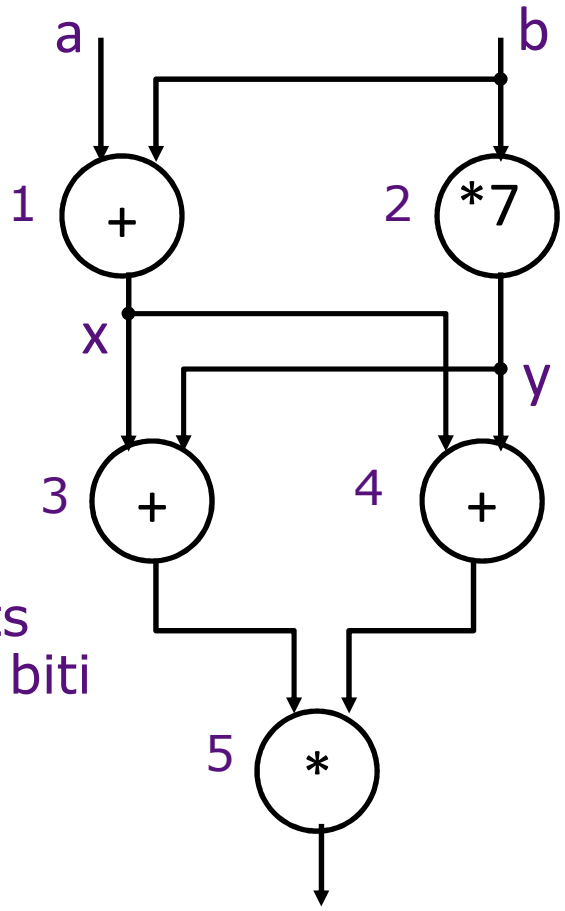
- Osnovna karakteristika: *single-assignment rule*
  - **Promenljiva se može pojaviti samo jednom na levoj strani izraza unutar programa u kome je aktivna. Kompajler kontroliše pravilo – funkcionalno programiranje. Po ostalim osobinama liče na PASCAL (izbegavanje pokazivača)**
- Primeri: VAL, Id, LUCID
- Prevodioci prevode program u dataflow orijentisani graf sa definisanim simbolima – dataflow operatorima.
- Kompajler takođe definiše data tokene koji inicijalno imaju spremne validne podatke (data ready)
- Tok tokena prilikom izvršavanja čini da neki od čvorova (instrukcija) postaju enabled (data ready) i zatim dolazi do izvršavanja (fires).

Statičke Dataflow Mašine:

Template-i instrukcija

	Opcode	Destination 1	Destination 2	Operand 1	Operand 2
1	+	3L	4L		
2	*	3R	4R		
3	−	5L			
4	+	5R			
5	*	out			

Presence bits
<=> Ready biti



Svaka grana u grafu ima mesto u Template-ima instrukcija, pa samim tim u programu.

Vezivanje instrukcija, argumenata i destinacija

- Instrukcije se asinhrono izvršavaju, pa ne sme da se pravi nikakva pretpostavka o redosledu izvršavanja data ready instrukcija
- Broj destinacija je promenljiv i nije limitiran na 2, pa mora postojati lista destinacija (instrukcija/port)
- Oslonjeno na single assignement rule, pa se svaki čvor samo jednom izvršava
- petlje?

Static dataflow mašine

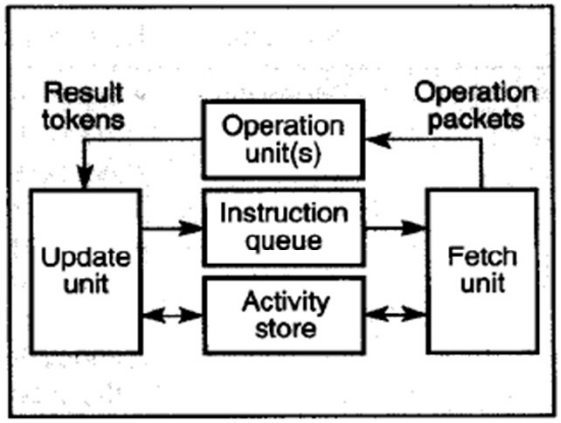


Figure 1. The basic organization of the static dataflow model.

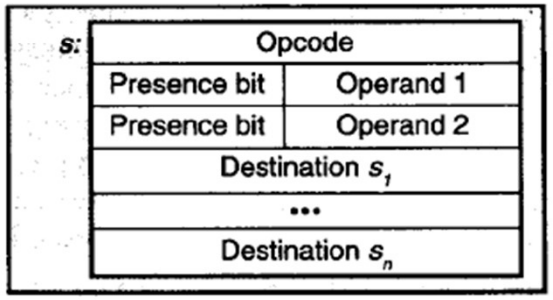


Figure 2. An instruction template for the static dataflow model.

Activity store sadrži sve instrukcije date kao instrukcijske template

Instrukcijski red sadrži pokazivače (memorijske adrese) instrukcijskih template koji su data ready

Fetch jedinica ima ulogu da spremne template instrukcije iz Activity store pošalje kao operacione pakete do operacionih (izvršnih jedinica).

Operacione jedinice nakon izvršavanja instrukcije generišu data token koji *Update jedinica* na osnovu destinacija u template-u instrukcija prosleđuje do destinacionih instrukcija u *activity store*.

Ažuriraju se vrednosti polja odgovarajućih operand polja i presence bita relevantnih instrukcija

Activity store vraća adrese instrukcijskih template-a koji su postali data ready preko *Update jedinice* u *Instrukcijski red*

Petlje static dataflow

- Problemi:
  - Može da se javi više tokena na istoj grani i nema načina da se razlikuju prema iteraciji
  - Redosled tokena ne mora da bude kao redosled izvršavanja iteracija
  - Zbog asinhronog izvršavanja, potreban je FIFO neograničene veličine
  - Mora se ograničiti da na jednoj grani može da bude samo jedan data token

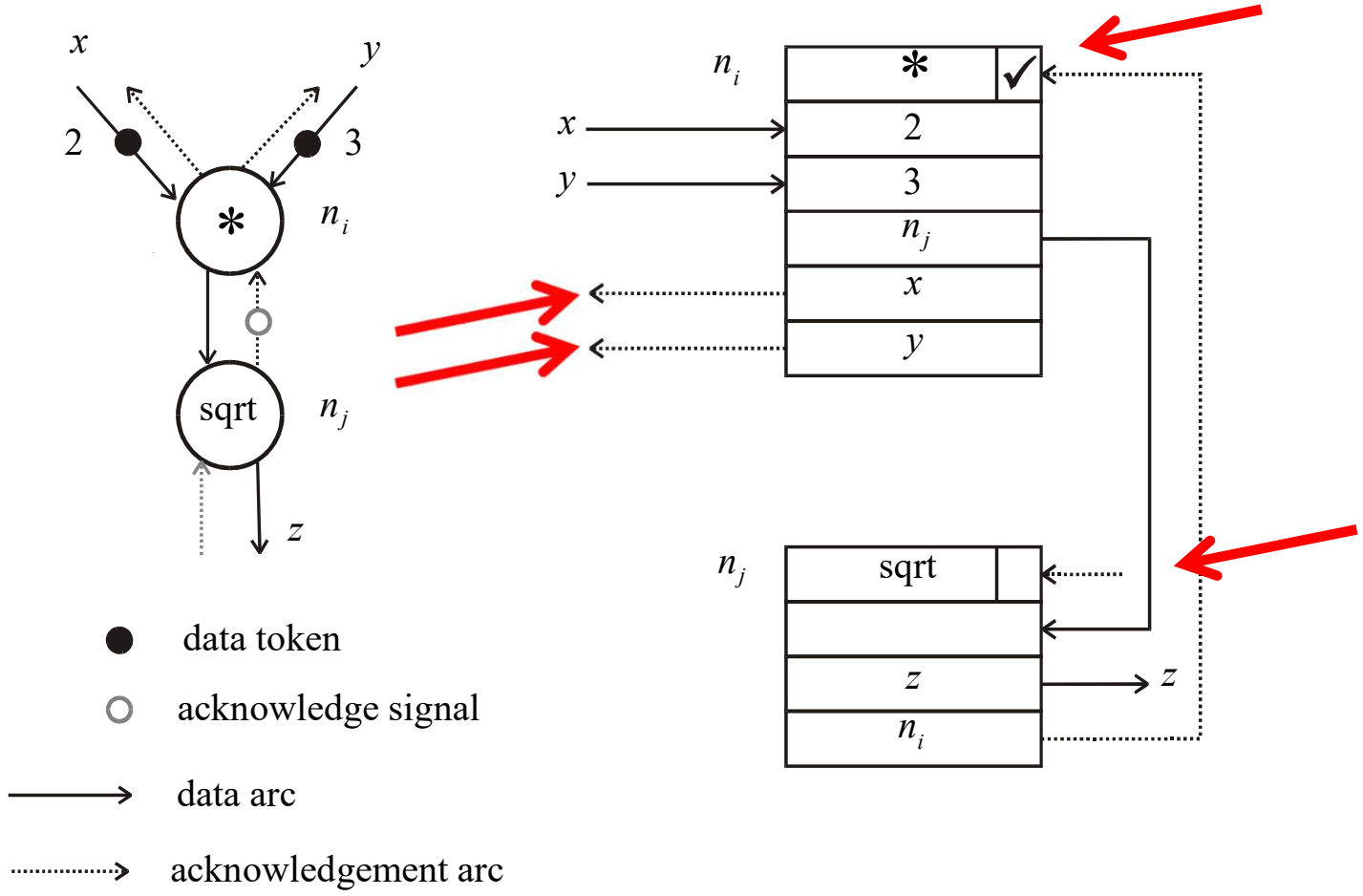
Acknowledgement signali

- *Proširenje zahteva da neka instrukcija može u izvršenje:*
  - **Omogućeni (data ready, enabled) čvor (jednostavna operacija, instrukcija) može da se izvrši samo ako nema nijednog tokena na bilo kojoj njegovoj izlaznoj grani (ovde se računaju sve grane nakon Fork)**.
- Ovo ograničenje se implementira preko signala potvrde (*acknowledge signals)* koji su u osnovi dodatni tokeni ka čvoru proizvođaču. Oni putuju preko dodatnih grana od svih čvorova potrošača generisanog tokena ka čvoru proizvođaču data tokena.
- To menja definiciju kada je jednostavna operacije omogućena (enabled), jer se traži i da su stigli signali potvrde od svih potrošača izlaznog data tokena prethodno završene jednostavne operacije

Rezultat ?

- Pravilo izvršavanja ostaje isto pri tako izmenjenom pravilu omogućavanja
- Smanjen paralelizam, jer za DoAll petlje je dozvoljeno da se preklapa mali broj iteracija i za neograničene resurse mašine
- Smanjena brzina izvršavanja zbog propagacije acknowledgement signala
- Broj tokena dupliran!!!

Dataflow graph i Activity template

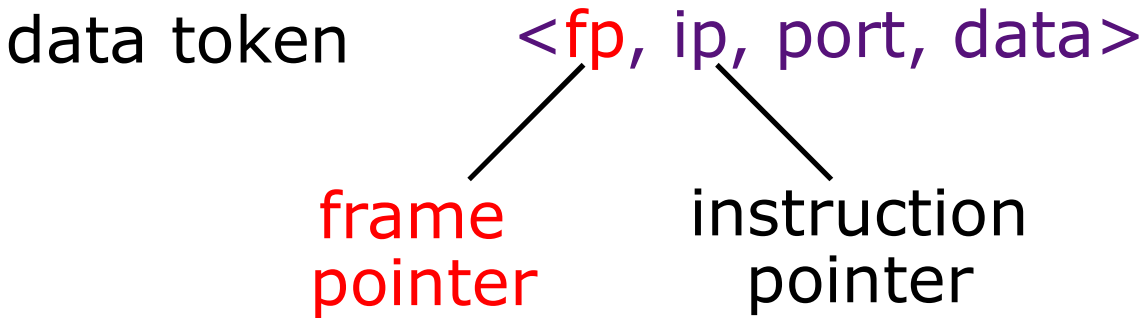


Osnovne mane

- Reentrantne rutine?
- Pozivi procedura – nema podrške
- Rekurzije – nema podrške
- DoAll ograničava na neku mešavinu delimičnog polu razmotavanja i softverske protočnosti sa ograničenim nivoom softverske protočnosti LSP

Dinamičke Dataflow Arhitekture

- Ideja – dinamički alocirati template, odnosno frame instrukcije da se podrži svaka iteracija petlje i svaki poziv procedure
  - Potrebno je detektovati završetak da se dealociraju frame – ovi instrukcija
- Kôd više petlji se može pomešati ako odvojimo pamćenje koda i data tokena. Pritom data token dobija novo polje frame pointer, sa ciljem da se jedinstveno označe podaci iz istih iteracija ili poziva procedura.

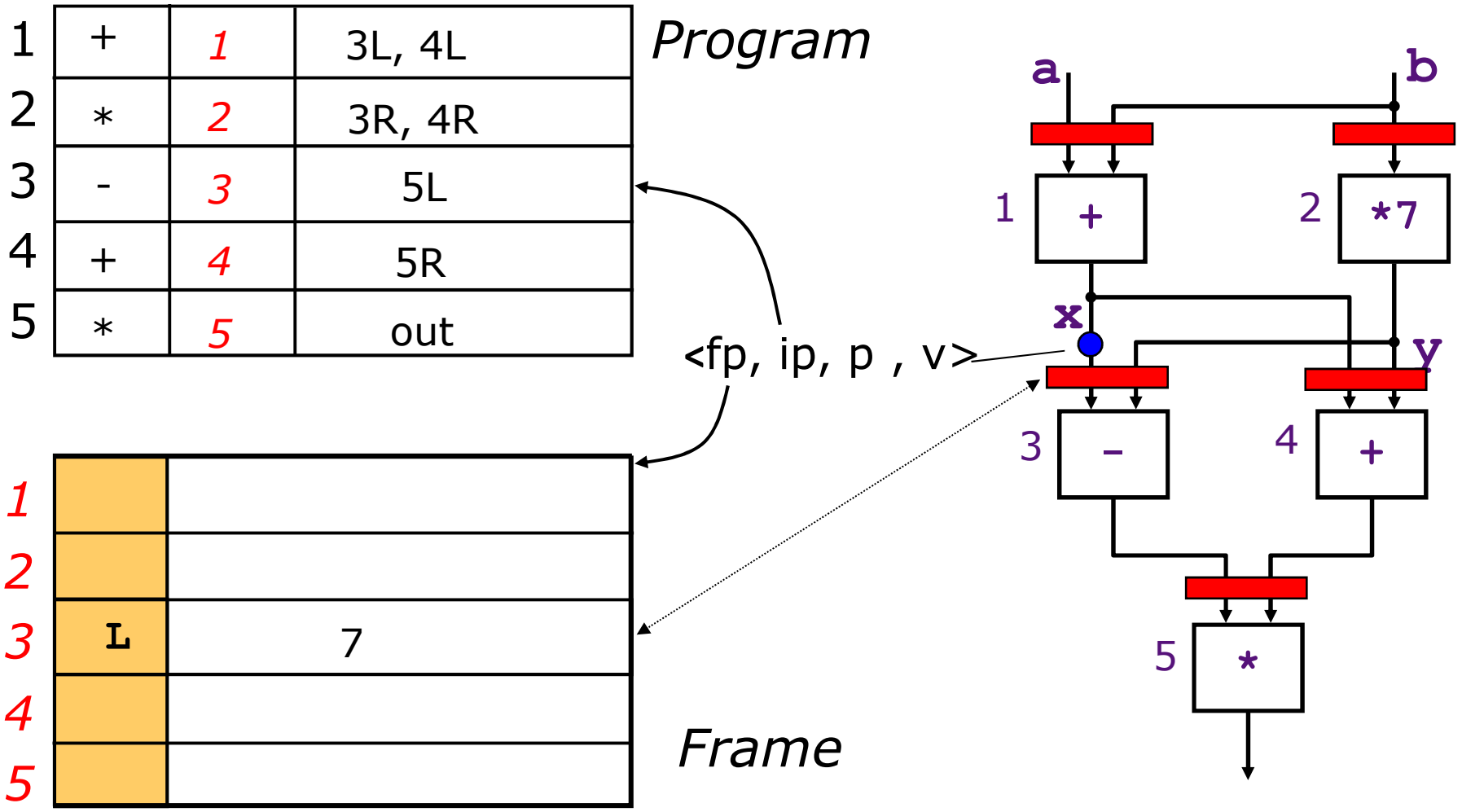


Paralelizam

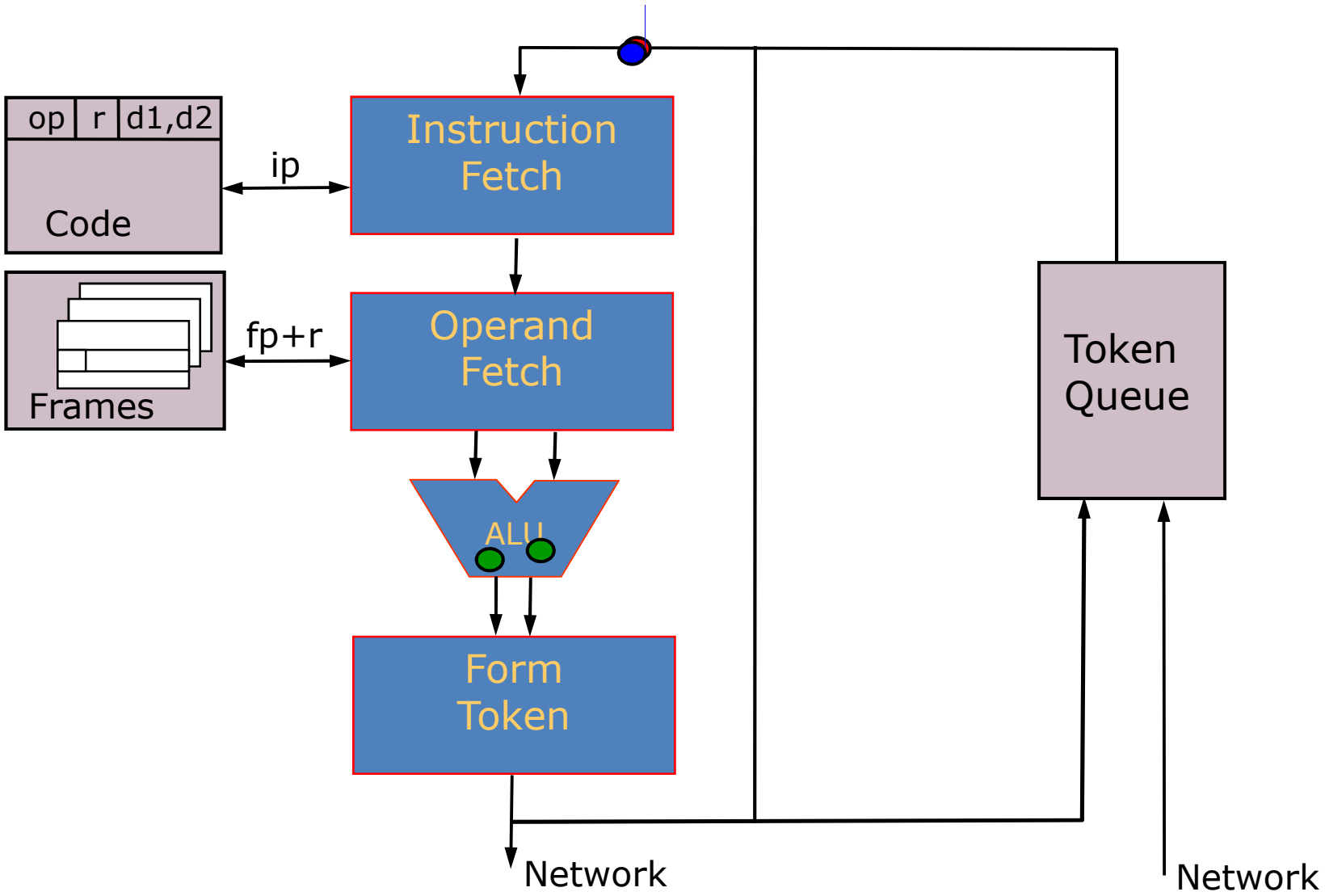
- Iteracije petlji ili poziv podprograma nemaju organičenje u mašini da se izvršavaju u paraleli kao zasebne instance reentrantnog podgrafa.
- Replikacija je samo konceptualna i ograničena brojem bita u frame pointeru.
- Svaki data token ima integralni tag koji čine:
  - Adresa instrukcije i porta za koju je namenjen
  - Kontekstna informacija – frame pointer
- Svaka grana se posmatra kao torba koja može da sadrži proizvoljan broj tokena sa različitim tagovima.
- Novo pravilo kada je omogućeno izvršavanje čvora je:

  Čvor je omogućen kada tokeni sa identičnim tagovima prisutni na svim ulaznim granama.

Frame kod Dynamic Dataflow

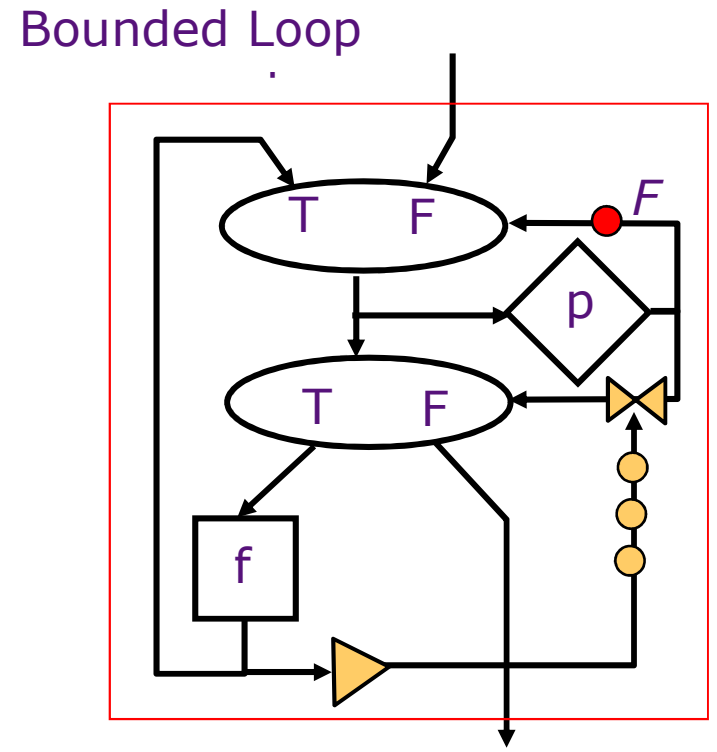


Monsoon Processor



Petlja

Postoji potreba za praćenjem frame-ova vezanih za svaku iteraciju

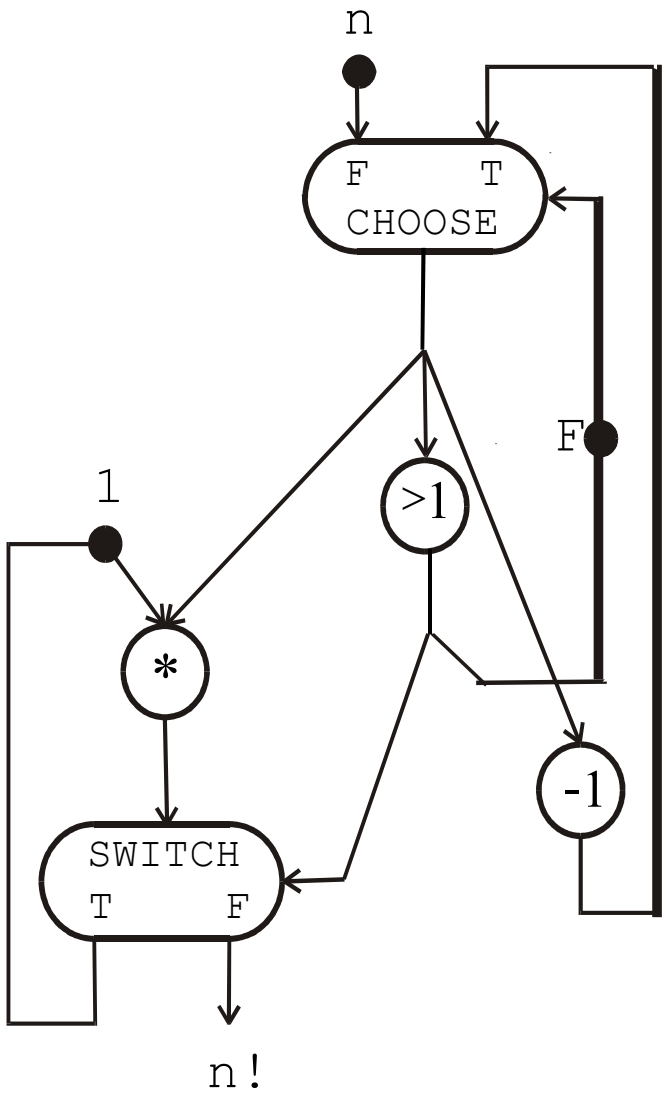


Primer u jeziku za funkcionalno programiranje Id

- Id programski segment
- izračunava n!

  prirodnog broja n

```
( initial j <- n; k <- 1
  while j > 1 do
   new j <- j - 1;
   new k <- k * j;
  return k )
```



Kontrola Paralelizma

- Problem: Mnogo iteracija može da bude prisutno u mašini u nekom trenutku
  - 100K iteracija na 256 procesorskoj mašini (raspad dela za token matching)
  - Može da dođe do deadlock-a
- Rešenje: Ograničiti koliko iteracija može istovremeno da bude u mašini u istom trenutku.
  - Zahteva promene u dataflow grafu petlje time što se zakoči generisanje novih tokena, kada je broj iteracija veći od N

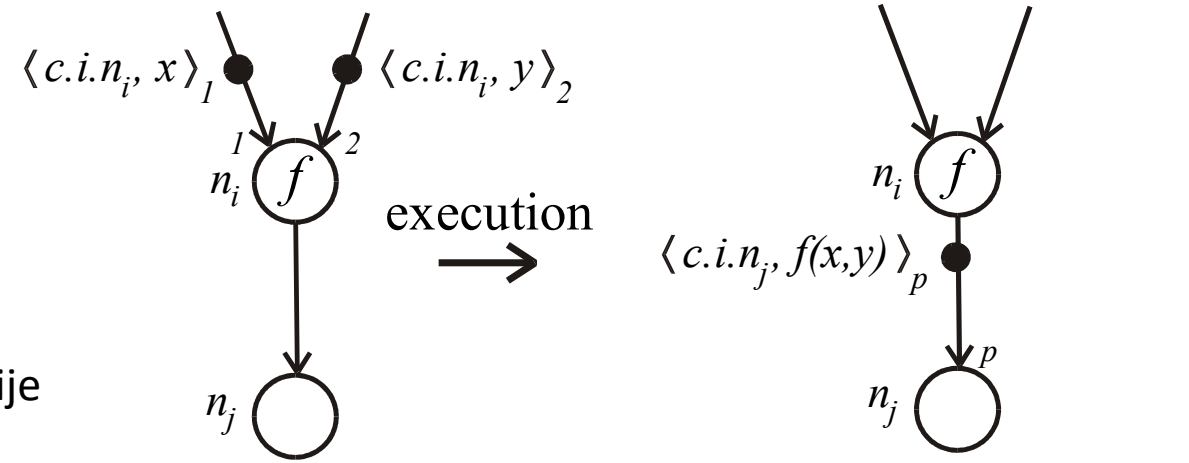
Notacija za dinamičku data flow mašinu

- Ako čvor n_i izvršava dijadnu funkciju f i ako je port p od n_j destinacija rezultata n_i, tada važi

$$in : \{\langle c.i.n_i, x\rangle_1, \langle c.i.n_i, y\rangle_2\} \qquad out : \{\langle c.i.n_j, f(x,y)\rangle_p\}$$

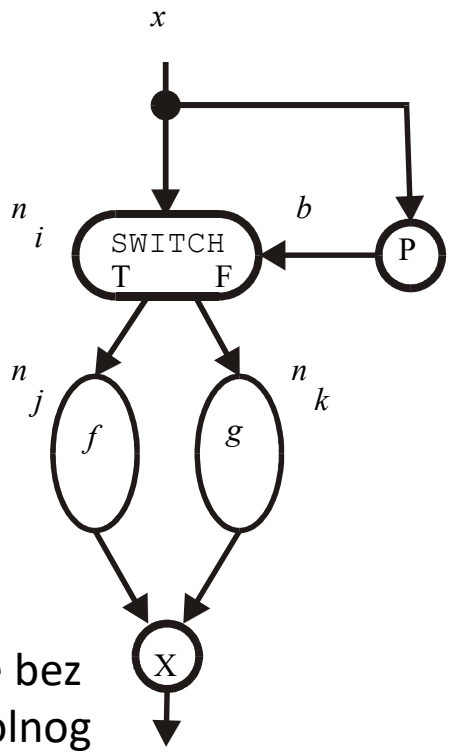
$\langle c.i.n, data\rangle_p$

c – frame
i – iteracija
n – adresa instrukcije
data – podatak
$_p$ - port



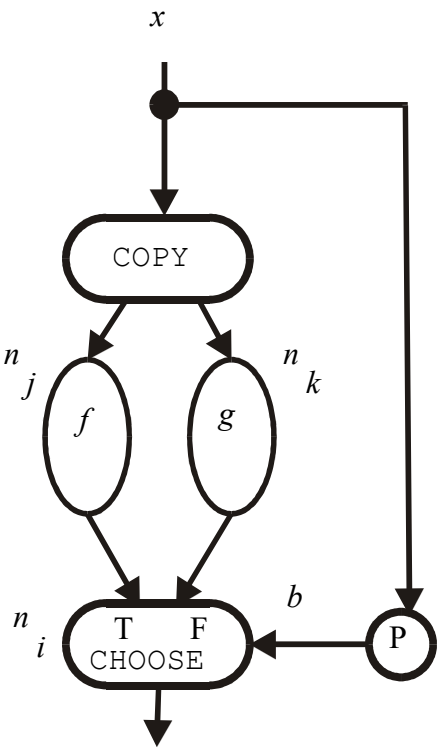
Grananje u dinamičkoj data flow mašini

$$in : \left\{ \left\langle c.i.n_i, x \right\rangle_{data} ; \left\langle c.i.n_i ; b \right\rangle_{control} \right\} \qquad out : \begin{cases} \left\{ \left\langle c.i.n_j, x \right\rangle \right\} if & b = T \\ \left\{ \left\langle c.i.n_k, x \right\rangle \right\} if & b = F \end{cases}$$



Merge bez Kontrolnog ulaza

Branch



Speculative branch evaluation

Spekulativno izvršavanje na dinamičkoj data flow mašini

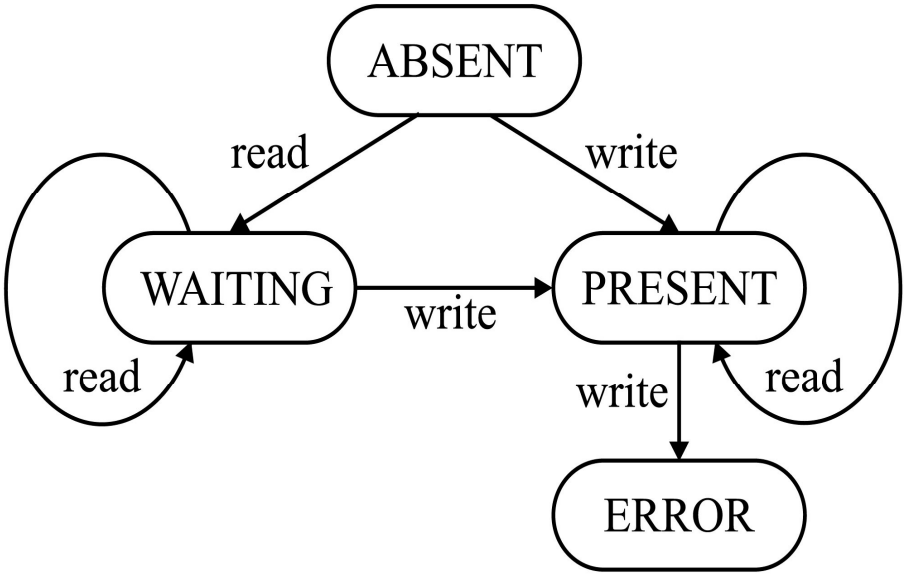
- Nema kontrolom pokretanog izvršavanja, pa nema ni spekulativnosti po kontroli
- Spekulativno grananje izvršava obe grane i na kraju se odlučuje iz koje grane se puštaju rezultati
- Eksplozija varijanti ukoliko se ide u dublju spekulativnost
- Spekulativno izvršavanje je neisplativo!

Nizovi i I-strukture

- status svakog elementa I-strukture može biti:
  - *prisutan*: element može da se čita, ali ne da se upisuje,
  - *odsutan*: pokušaj čitanja mora da bude odložen, ali je dozvoljen jedini upis,
  - *čekanje*: bar jedan zahtev za čitanje postoji za element I-strukture koji je na čekanju.
- Nakon što je element postao *definisan* (inicijalizovan, dodeljena vrednost; može se desiti samo jednom), sva *odložena čitanja*, koja se čuvaju u asocijativnom redu, postaju odmah zadovoljena.
- I-strukture čine strukture podataka upotrebljivim na nivou elemenata, pre nego što je cela struktura definisana.

I-strukture dijagram stanja

- Status svakog elementa I-strukture može biti:
  - *present*: može se čitati,
  - *absent*: zahtev za čitanje se mora zakasniti, a upis je dozvoljen,
  - *waiting*: postoji bar jedan read zahtev koji je zakašnjen.



I-strukture elementarne operacije

- Elementarne operacije definisane za I-strukture:
  - *allocate*: rezerviše definisani broj elemenata za novu I-strukturu,
  - *I-fetch*: čita sadržaj određenog elementa I-strukture (ili zakašnjava čitanje)
  - *I-store*: upisuje vrednost u element I-strukture, a ako element nije prazan javlja grešku.
- *I-fetch* instrukcije su sa memorijskim operacijama sa razdeljenim fazama :
  Zahtev za čitanje elementa je nezavisan u vremenu od vremena odgovora (čeka se da element postane data ready).

Data Flow Prednosti i mane

- Prednosti
  - Vrlo dobre mašine za korišćenje iregularnog paralelizma
  - Samo prave zavisnosti ograničavaju obradu
- Mane
  - Oslanja se na zavisnosti određene u vreme prevođenja, pa nekada dodaju zavisnosti i kad ih nema.
  - Veoma težak debugging (nema preciznog stanja)
    - Prekidi i obrada izuzetaka se teško izvode(u kom trenutku nastaje?)

Mane (1)

- Primena dinamičkih struktura podataka teška u čistom data flow modelu
- Nemogućnost obrade kod prevelikog paralelizma? (Kontrola Paralelizma potrebna)
- Ogroman posao vezan za uparivanje tagova (asocijativne memorije za ogroman broj instrukcija i tokena ili kombinacija sa interkonekcionim mrežama)
- Veliko zauzeće memorije za tokene podataka
- Instrukcijski ciklus je neefikasan jer postoji veliko kašnjenje u sprovođenju data tokena između zavisnih instrukcija

Mane (2)

- Ne koristi se lokalnost u memoriji
- Jezici za programiranje nemaju pokazivače
- Spekulativno izvršavanje po kontroli praktično ne postoji, pa se ne koristi taj izvor paralelizma
- Dodatni ciklusi zbog dodatih select i merge operatora za grananja
- Operacije nad vrstama i kolonama niza se uvek svode na nezavisne operacije nad elementima I-struktura
- Nemogućnost da se uspostave dugački pipeline-i instrukcija

Ima li nade za data flow?

- Hardver sa umapiranim data flow grafom algoritma npr. brza furijeova transformacija - FFT
- Takvi algoritmi u ruterima, grafičkim procesorima, ...
- Unutrašnje petlje DoAll tipa se kompajliraju pomoću silicon kompajlera i ulaze u programabilnu logiku sa idejom – svakog ciklusa jedna iteracija (kao hardverizovan software pipelining) – problem kako dovoljno brzo dostaviti ulaze i primiti izlaze – vektorski registri?

Ima li nade za data flow (1)?

- Kombinovati dobro iz data flow i control flow – dinamički odrediti za deo instrukcija DDG i sa tim raditi kao ugrađenom data flow mašinom sa ograničenim brojem operacija
  - Iskoristiti memorijski paralelizam preko regularnih rasporeda struktura u interleaved memoriji
  - Iskoristiti spekulativnost po kontroli u toj kombinaciji