

ELEKTROTEHNIČKI FAKULTET, UNIVERZITET U BEOGRADU
PARALELIZAM U ARHITEKTURI RAČUNARA I PROGRAMSKIM PREVODIOCIMA

VEKTORSKI PROCESORI

doc. dr Maja Vukasović
as. ms Nikola Stanković

Vektorski procesori

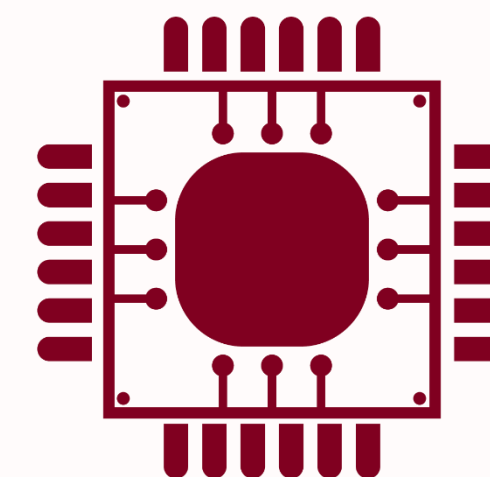
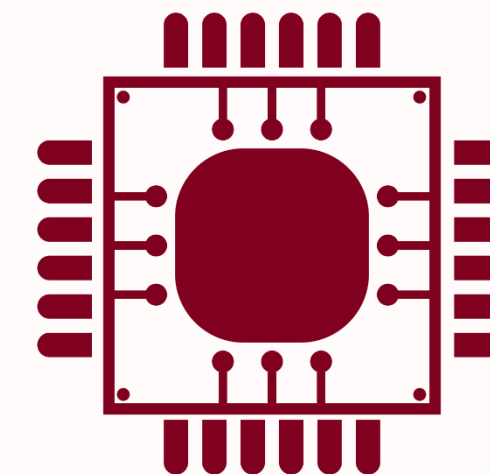
- **Skalarni procesori** – operandi instrukcija su skalarni podaci.



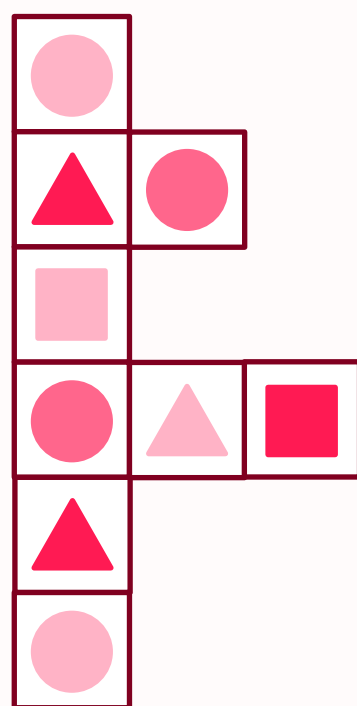
- **Vektorski procesori** – operandi instrukcija su vektori podataka umesto pojedinačnih vrednosti.



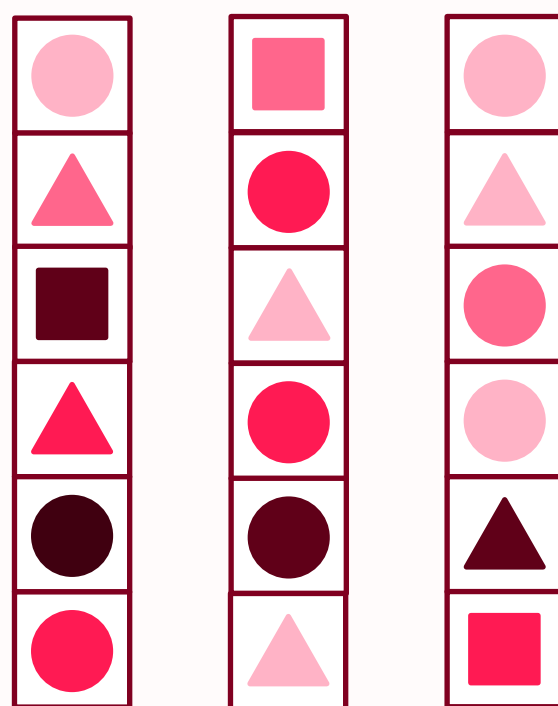
- Jednom instrukcijom se specificira da istu operaciju treba primeniti na sve elemente vektora (niza).
- Koristi se i termin SIMD (*Single Instruction Multiple Data*).



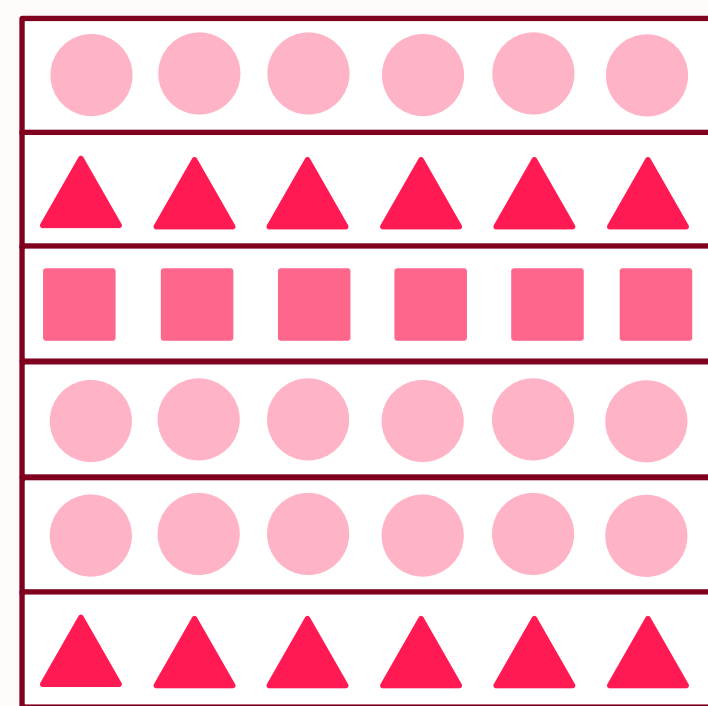
Nivoi paralelizma



Instrukcijski nivo
paralelizma



Paralelizam na
nivou niti



Paralelizam podataka u
vektorskim procesorima

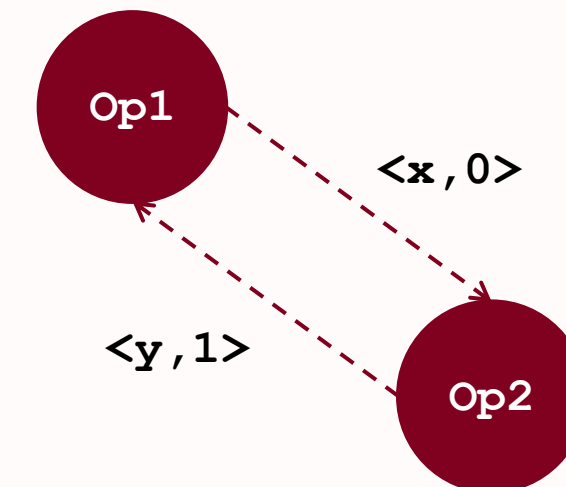
Vektorske operacije (1)

```
DO 100 I=1,N  
Op1: A(I) = B(I) + C(I)  
Op2: B(I) = 2 * A(I+1)  
100 CONTINUE
```

Razmotana petlja

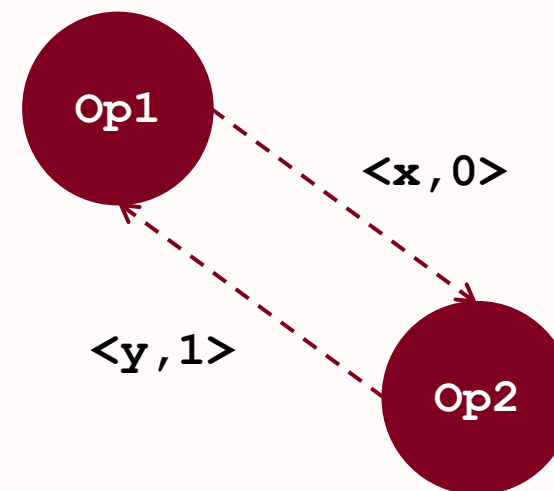
```
A(1) = B(1) + C(1)  
B(1) = 2 * A(2)  
A(2) = B(2) + C(2)  
B(2) = 2 * A(3)  
...  
...
```

- DoAcross petlja zbog dve antizavisnosti na nivou nizova u memoriji:
 - $\langle x, 0 \rangle$ zbog $B(I)$
 - $\langle y, 1 \rangle$ zbog $A(I)$ - Op1 nove iteracije u odnosu na Op2 prethodne



Vektorske operacije (2)

```
DO 100 I=1,N  
Op1: A(I) = B(I) + C(I)  
Op2: B(I) = 2 * A(I+1)  
100 CONTINUE
```

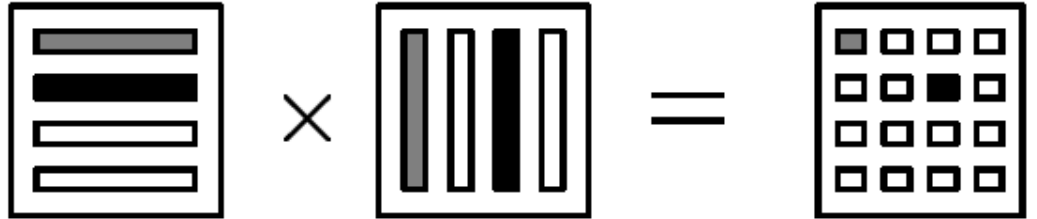


```
TEMP(1:N) = A(2:N+1)  
A(1:N) = B(1:N) + C(1:N)  
B(1:N) = 2 * TEMP(1:N)
```

- U okviru iteracije petlje na skalarnom procesoru postoje antizavisnosti – **preimenovanje na nivou vektora** (za $A(I)$) u **vektorskim registrima** prilikom realizacije na vektorskom procesoru omogućava vektorizaciju (nema više ciklusa u grafu petlje, pa postaje DoAll), a druga antizavisnost se poštuje redosledom (za $B(I)$).

Množenje matrica (1)

- Izračunavanje pojedinačnog elementa rezultujuće matrice u okviru sledeće petlje na skalarnom procesoru:

$$c_{ij} := \sum_{k=0}^{N-1} a_{ik} b_{kj}$$


```
; initialization
kLoop: ADD    Ri, stride-i      ; increment column of A
        ADD    Rj, stride-j      ; increment row of B
        LOAD   RA, A(Ri)         ; get value of matrix A row
        LOAD   RB, B(Rj)         ; get value of matrix B column
        FMPY   RA, RB            ; floating multiply
        FADD   RC, RA            ; floating add
        INC    Rk                ; increment k
        CMP    Rk, RN            ; at end of row x column?
        BNE    kLoop            ; no - repeat for R x C
        STORE  RC, C(r,c)        ; yes - store C element
        -----                ; continue with all rows/columns of C
```

Množenje matrica (2)

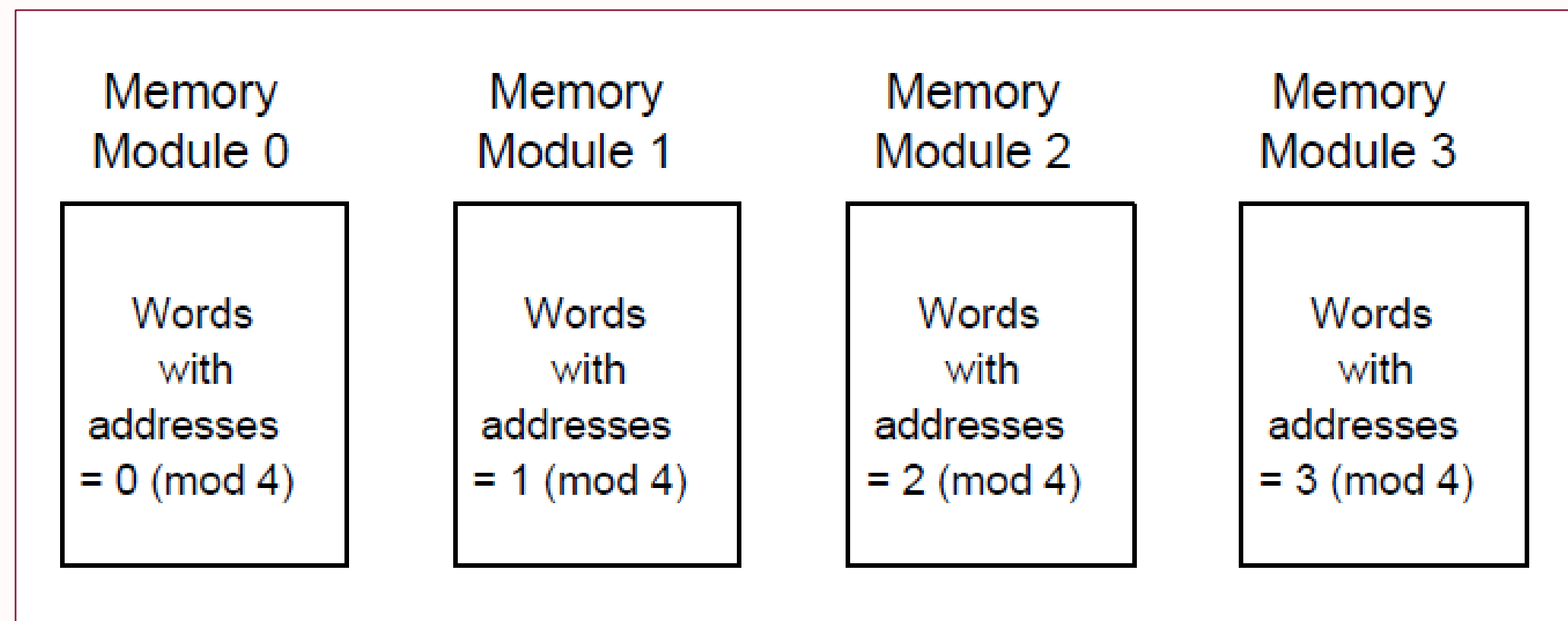
- U realizaciji ovog algoritma u vektorskim procesorima unutrašnja petlja zamenjena je sledećim operacijama:

```
VLOAD      V1,A(r),N, stride-i    ; vector load row of A with stride i
VLOAD      V2,B(c),N, stride-j    ; vector load column of B with stride j
VMPYADD     V1,V2,RC               ; vector multiply + add to C
STORE      RC,C(r,c)              ; store C element
```

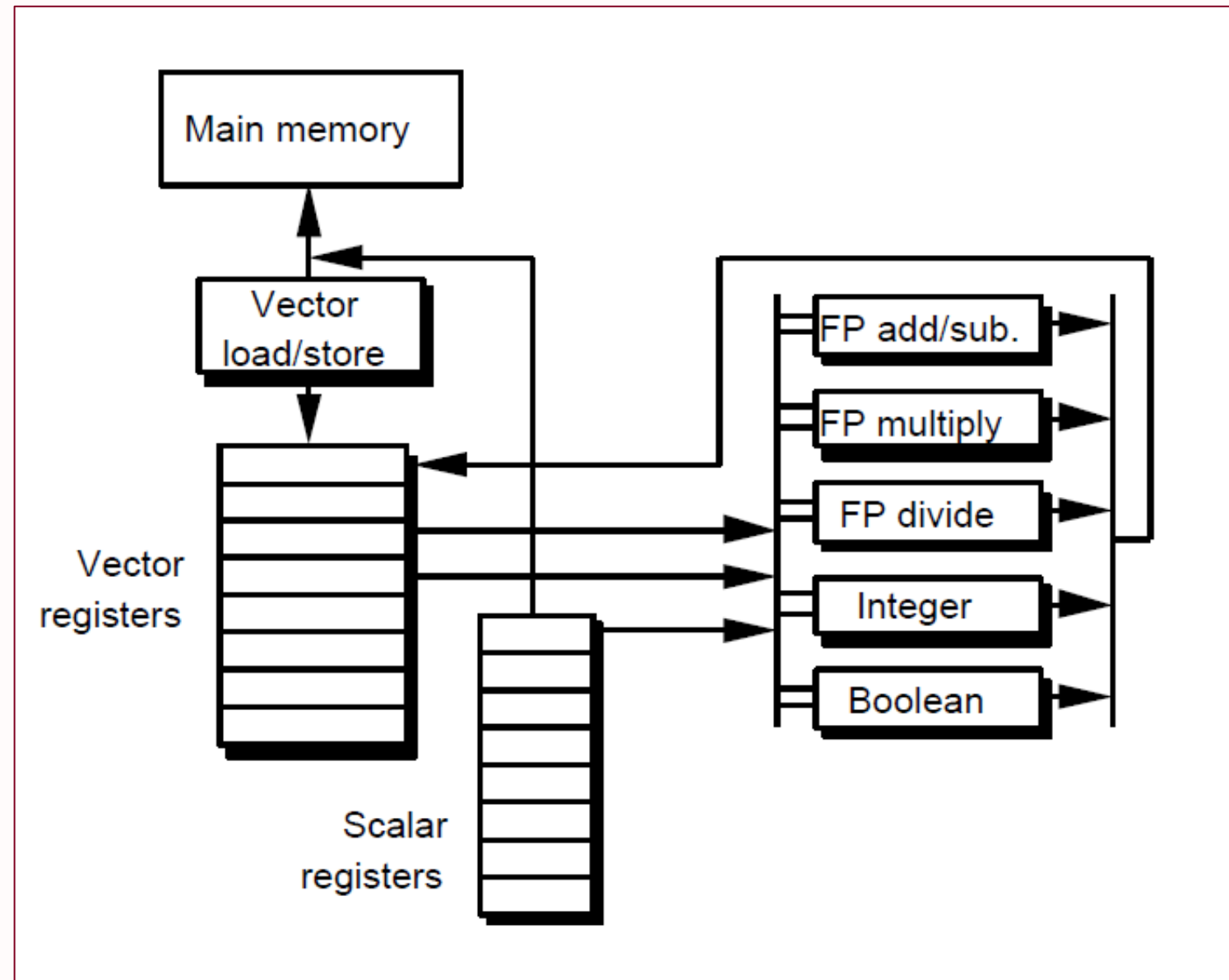
- Generisanje novog rezultata u svakom taktu.
- Udaljeni podaci u memoriji smeštaju se u kontinualne lokacije u okviru vektorskih registara.

Memorijske banke

- Organizacija memorije radi ubrzavanja sekvencijalnog pristupa memoriji.
- Čitanje i upis u različite memorijske banke može biti preklopljeno.



Arhitektura vektorskih procesora (1)



Varijanta arhitekture gde se svi izlazi pame u vektorskim registrima

Arhitektura vektorskih procesora (2)

- Podaci se učitavaju u vektorske registre – FIFO redovi.
- Instrukcije se izvršavaju tako što se niz podataka učitava iz memorije u vektorske registre, izvršava se vektorska operacija i nakon toga se podaci iz registra vraćaju u memoriju.
- Dostupne instrukcije zavise od komponenata u okviru procesora.
- U konkretnom slučaju moguće je sabiranje, oduzimanje, množenje i deljenje realnih brojeva, sabiranje i šiftovanje celih brojeva i logičke operacije.

Instrukcije vektorskih procesora

- Oba operanda instrukcije mogu biti vektori ili neki od njih može biti skalarna veličina. Na primer:

Instrukcija	Operandi	Opis	Operacija
ADDV.D	V1 , V2 , V3	Vektor + Vektor	V1 = V2 + V3
ADDSV.D	V1 , F0 , V2	Skalar + Vektor	V1 = F0 + V2
SUBV.D	V1 , V2 , V3	Vektor – Vektor	V1 = V2 – V3
SUBSV.D	V1 , F0 , V2	Skalar – Vektor	V1 = F0 – V2
SUBVS.D	V1 , V2 , F0	Vektor – Skalar	V1 = V2 – F0
LV	V1 , R1	Učitavanje u vektorski registar V1 od adrese R1	

Maskirajući registar

- Kontrolne zavisnosti smetaju izvršavanju u vektorskom procesoru i moraju biti eliminisane.
- Ovakav kod se, na primer, ne može prevesti u kod za vektorski procesor:

```
for i := 1 to n do  
    if A[i] != 0  
        then A[i] := A[i] - B[i];
```

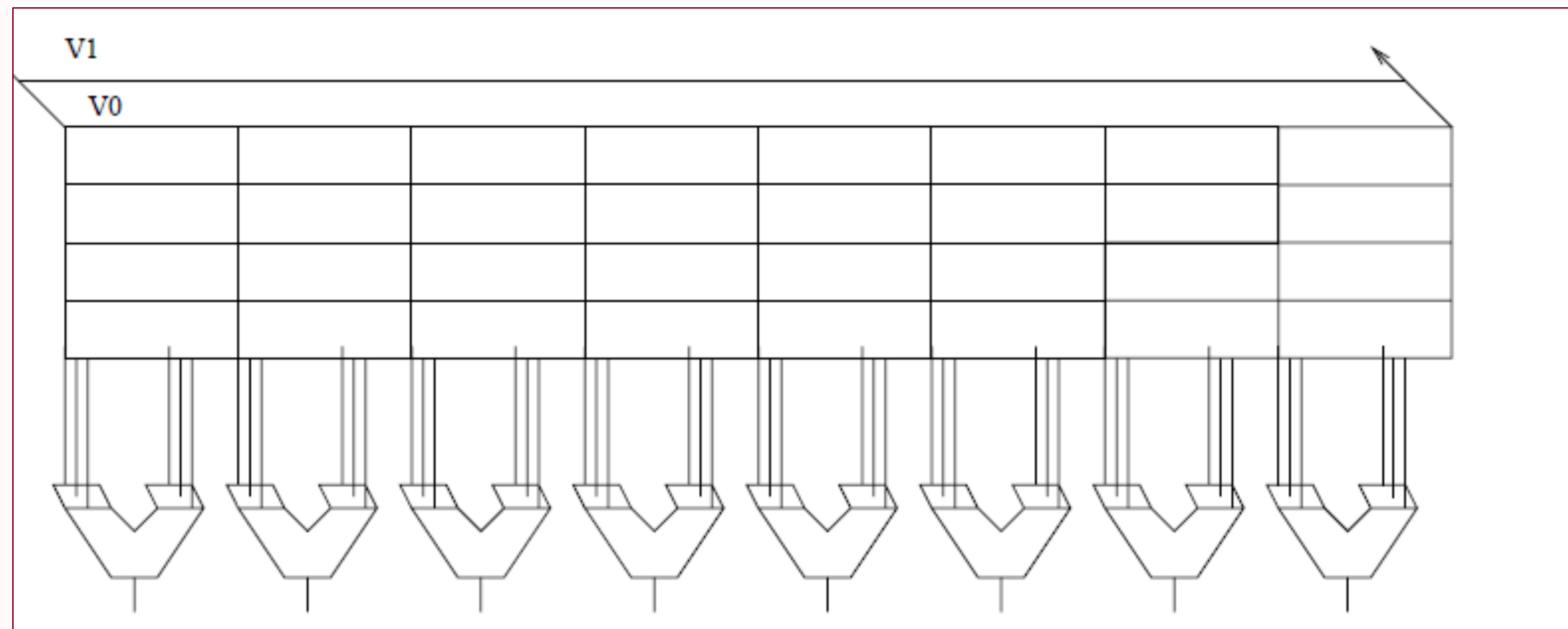
- Maskirajući registar služi da selektuje odgovarajuće elemente vektora na koje će operacija biti primenjena.

```
LV V1,Ra           ; load vector A into V1  
LV V2,Rb           ; load vector B  
L.D F0,#0         ; load FP zero into F0  
SNEVS.D V1,F0      ; sets VM(i) to 1 if V1(i)!=F0  
SUBV.D V1,V1,V2    ; subtract under vector mask  
SV Ra,V1          ; store the result in A
```

Pipes

- Implementacija zavisi od broja kanala.
- U konkretnom primeru, pošto postoji 8 kanala, 8 elemenata niza može biti obradjeno odjednom. Ako je dužina vektora 32, a jedan element se obradjuje u jednom ciklusu, ukupno je potrebno 4 ciklusa za ceo niz.

VLD V0, (V1)



Convoy (vector function chaining)

- Grupa vektorskih instrukcija koje mogu započeti u istom taktu (pomerene za kašnjenje zavisnosti po podacima) (*convoy*) - u okviru grupe ne postoje zavisnosti.

```
LV V1, Rx           ; load vector X
MULVS.D V2, V1, F0   ; vector-scalar multiply
LV V3, Ry           ; load vector Y
ADDV.D V4, V2, V3    ; vector-vector add
SV Ry, V4            ; store the result
```

- Ako registre posmatramo ne kao celinu već kao odvojene elemente onda, možemo rezultate nakon što postanu dostupni odmah proslediti narednoj instrukciji kroz pipeline.

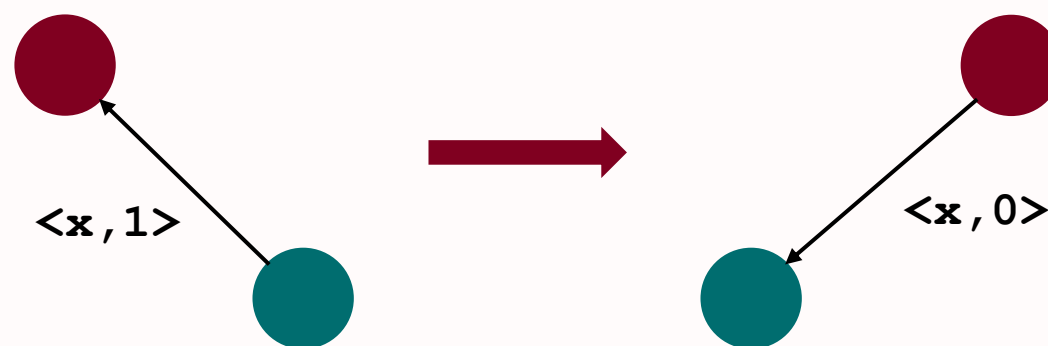
Convoy – primer (1)

- Primer transformisane doAll petlje koja može biti realizovana kao grupa (convoy) u vektorskom procesoru, samo ako se primeni softverska protočnost da bi prave zavisnosti po podacima bile unutar iste iteracije.

```
for (i = 1; i <= 100; i = i+1) {  
    ● A[i] = A[i] + B[i];  
    ● B[i+1] = C[i] + D[i];  
}
```



```
A[1] = A[1] + B[1];  
for (i = 1; i <= 99; i = i+1) {  
    ● B[i+1] = C[i] + D[i];  
    ● A[i+1] = A[i+1] + B[i+1];  
}  
B[101] = C[100] + D[100];
```



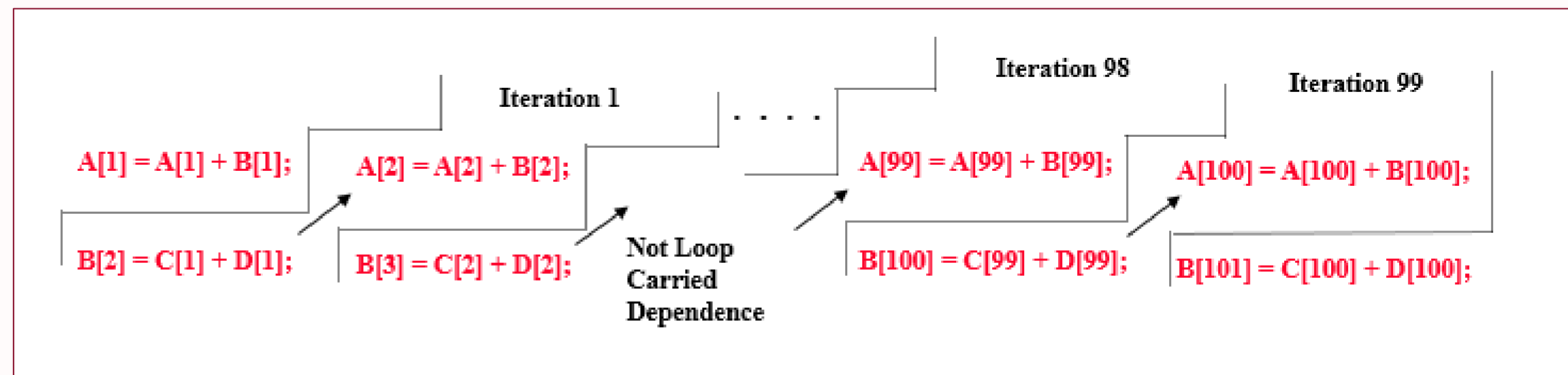
```
B(2:100) = C(1:99) + D(1:99)  
A(2:100) = A(2:100) + B(2:100)
```

Convoy – primer (2)

```
for (i = 1; i <= 100; i = i+1) {  
    A[i] = A[i] + B[i];  
    B[i+1] = C[i] + D[i];  
}
```



```
A[1] = A[1] + B[1];  
for (i = 1; i <= 99; i = i+1) {  
    B[i+1] = C[i] + D[i];  
    A[i+1] = A[i+1] + B[i+1];  
}  
B[101] = C[100] + D[100];
```



Maksimalna dužina vektora

- Maksimalni broj elemenata u vektoru koji se može obraditi u okviru jedne vektorske instrukcije ograničen je definisanom vrednošću u okviru procesora (*Maximum Vector Length*).
- Nekad je ukupan broj elemenata nekog niza poznat tek u vreme izvršavanja.
- Ukoliko je broj elemenata operanda instrukcije manji od maksimalne dužine, ceo niz može biti obradjen u okviru jedne instrukcije, u suprotnom radi se deoba niza na delove manje ili jednake maksimalnoj dužini vektora.