Uvod u paralelizam i vektorske instrukcije

(S. Ribarić, Arhitektura i organizacija računarskih sustava, str. 473-532)

- Oblici i razine paralelizma
- Flynnova taksonomija paralelnih arhitektura
- Vektorski procesori i instrukcije
- Vektorske ekstenzije arhitekture x86

Paralelna računala: proširenje Von Neumannovog modela koje omogućava usporedno provođenje različith koraka obrade.

- sklopovlje je pogodno za paralelnu obradu
- međutim, rješenja naših problema najlakše je slijedno formulirati

Oblici raspoloživog paralelizma u računarskim problemima:

- raspoloživi funkcijski paralelizam – *različiti* zadatci mogu se izvoditi usporedno (npr. učitavanje i obrada podataka)
- raspoloživi podatkovni paralelizam – *isti* zadatci mogu se izvoditi na različitim podatcima
  (npr. tražimo zadani objekt u različitim dijelovima slike)

Četiri razine (zrnatosti) raspoloživog funkcijskog paralelizma:

- paralelizam na razini instrukcija (ILP) – fino zrnati
  - protočnost, superskalarnost
- paralelizam na razini programskih petlji – srednje zrnati
  - višedretvenost, višejezgrenost (+jezična podrška)
- paralelizam na razini funkcija – srednje zrnati
  - višedretvenost, višejezgrenost (+jezična podrška)
- paralelizam na razini programa – grubo zrnati
  - višejezgrenost

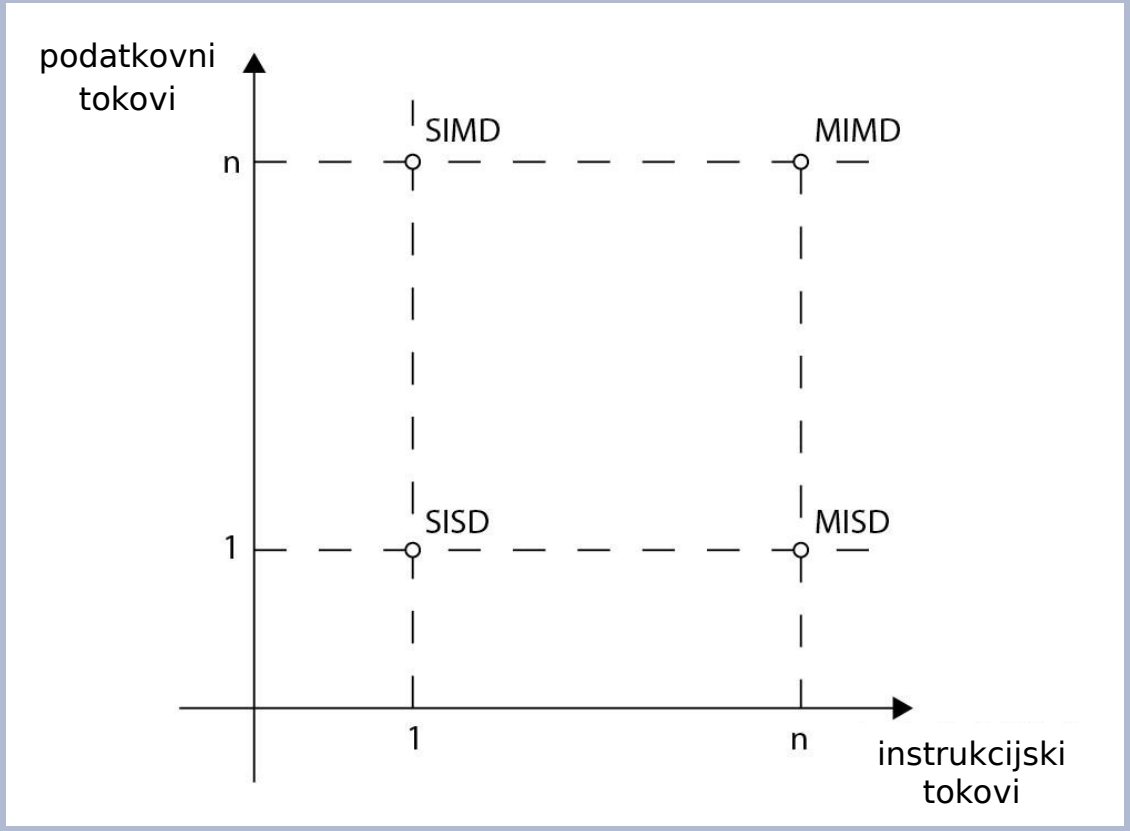
Podatkovni paralelizam može se iskoristiti na dva načina:

- izravno uporabom arhitektura procesora koje podržavaju paralelne operacije na podatkovnim elementima
  - vektorska računala i instrukcije: SIMD, SIMT
  - SIMD: ***S**ingle **I**nstruction Stream **M**ultiple **D**ata Stream*
  - SIMT: ***S**ingle **I**nstruction, **M**ultiple **Threads***
  - SMT: to je nešto sasvim različito!
- pretvorbom raspoloživog podatkovnog paralelizma u funkcijski
  - viši programski jezik označava paralelno izvodljive operacije na podatkovnim elementima (OpenMP)

Računalni sustavi tipa SIMD iskorištavaju podatkovni paralelizam djelovanjem na vektorima s višedimenzionalnim poljima podataka.

Na primjer, instrukcija `mulps zmm1, zmm1,zmm2` (AVX-512) započinje množenje 16 parova operanada i to s latencijom od 4-6 perioda signala vremenskog vođenja.

Flynnova taksonomija računalnih sustava



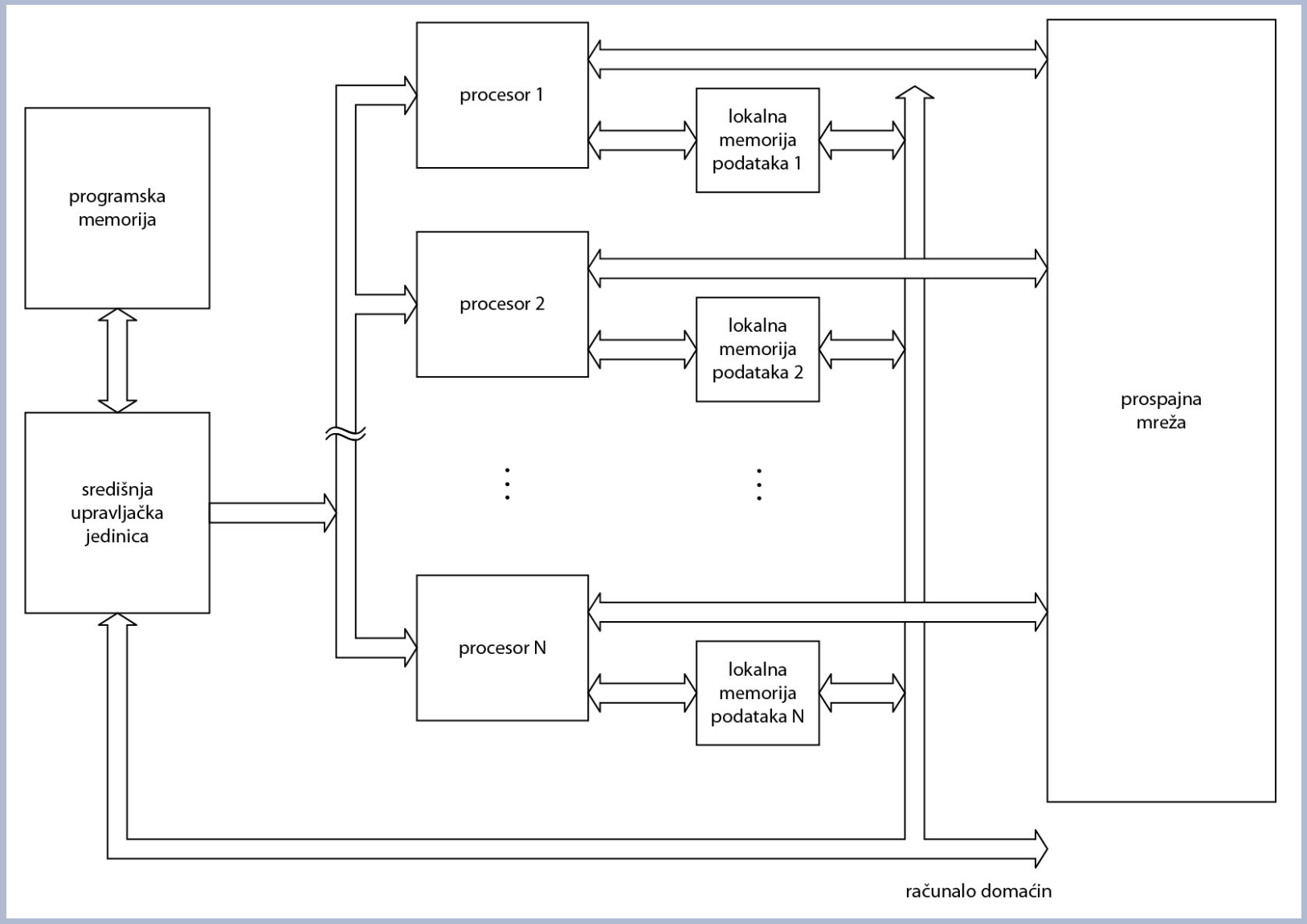
Prema Flynnovoj klasifikaciji, arhitekture paralelnih računalnih sustava su: MISD, SIMD i MIMD

- SIMD - procesori s vektorskim instrukcijama
- MIMD – multiprocesorski sustavi

Osnovna značajka SIMD arhitekture je istodobno izvođenje iste instrukcije na većem broju procesnih jedinica koji djeluju na različitim, višestrukim tokovima podataka.

Takva računala namijenjena su rješavanju problema s visokim stupnjem podatkovnog paralelizma:

- matrično množenje
- obrada jezika, govora, slika, 3D slika, oblaka točaka



Organizacija višeprocesorskog SIMD računarskog sustava

Vektorski procesori i instrukcije

Jedan od najdjelotvornijih načina iskorištavanja podatkovnog paralelizma postiže se u računalnim sustavima arhitekture SIMD koji se temelje na *vektorskom procesoru*
(engl. *vector processor*)

Vektorski procesor obavlja aritmetičke i logičke operacije na operandima koji su vektori.

Primjer

Razmotrimo računanje sume dvaju 64-dimenzionalnih vektora **x** i **y**. Rezultat je vektor **w**:

$$\mathbf{w} = \mathbf{x} + \mathbf{y}.$$

U "običnom" jednoprocesorskom sustavu ta bi se operacija izvela na temelju programskog odsječka:

```
for i = 1 to 64
    w(i) := x(i) + y(i);
```

U vektorskom procesoru gornja bi se operacija izvela samo *jednom vektorskom instrukcijom*, odnosno instrukcijom tipa *vektor-vektor* kojoj su operandi dva 64-dimenzionalna vektora, a rezultat, koji se dobiva u vektorskoj aritmetičko-logičkoj jedinici (vektorska ALU), također je 64-dimenzionalni vektor.

Vektorska ALU može istodobno zbrojiti/izmnožiti veći broj odgovarajućih komponenata obaju vektora.

Svaki od vektora, koji predstavlja
operand u vektorskoj instrukciji, smješten je u vektorski registar, npr. V_i, odnosno V_j, a rezultat se smješta u vektorski registar V_k.

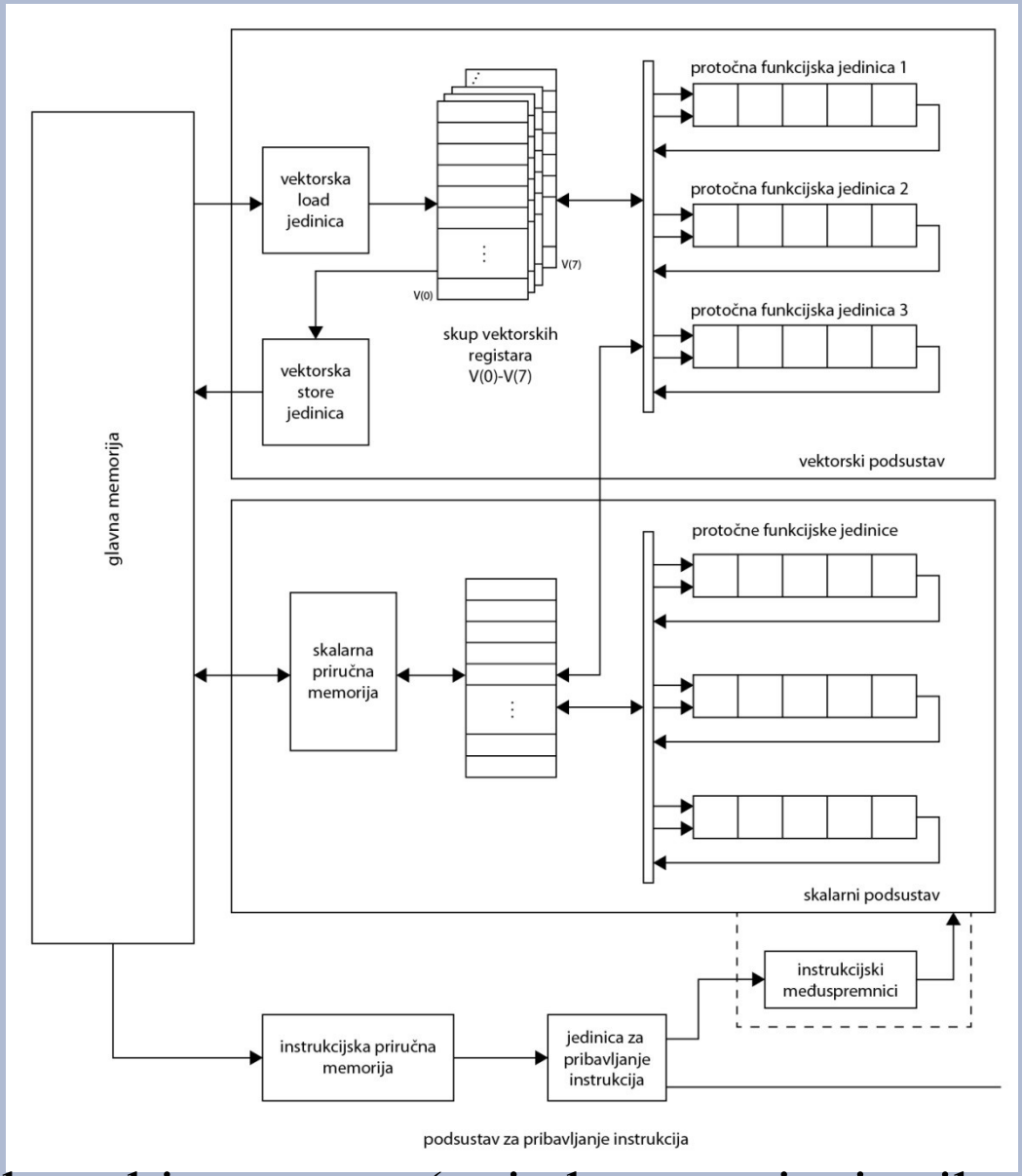
Vektorska instrukcija specificira veliku količinu posla i jednakovrijedna je, u potonjem slučaju, cijeloj programskoj petlji.

Uporaba vektorske ALU u kojoj se istodobno izvode operacije nad svim komponentama vektora u vektorskim je procesorima ipak rijetka.

Umjesto, na primjer, n vektorskih množila
(koja bi istodobno generirala sve produkte),
u praksi se koristi protočno množilo
s relativno velikim brojem protočnih segmenata.

Svaki parcijalni rezultat (umnožak dviju komponenata)
dobivamo u odgovarajućem koraku obrade
čije je trajanje određeno trajanjem obrade
u jednom protočnom segmentu.

Razlozi takva rješenja nisu tehnološka ograničenja,
već jednostavnija izvedba.



Vektorski procesor (pojednostavnjeni prikaz)

Tipovi vektorskih instrukcija

i) instrukcije vektor-vektor:

$$f_1: V_i \rightarrow V_k$$

$$f_{2:}\ V_i \times V_j \rightarrow V_k,$$

V_i i V_j označavaju izvorišni, a V_k odredišni vektorski registar;

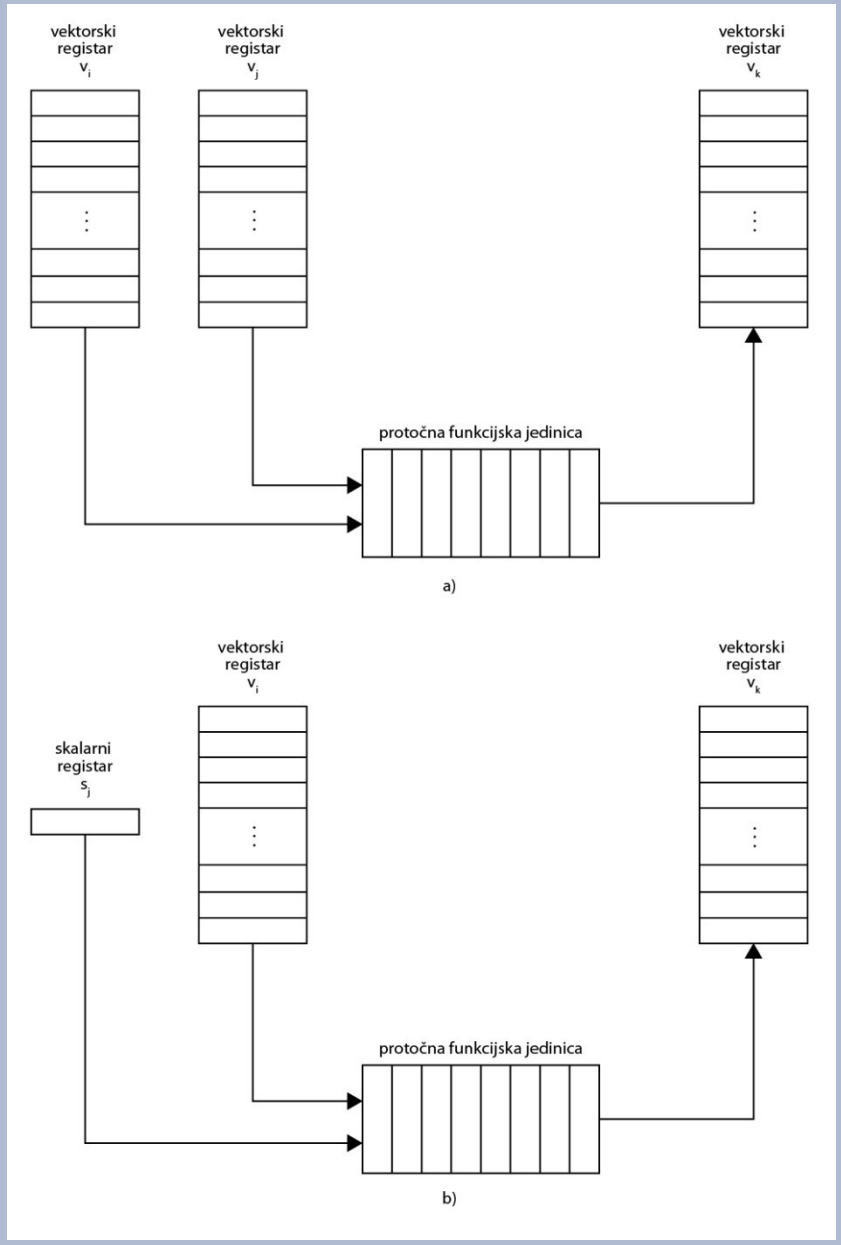
ii) instrukcije vektor-skalar (s_j označava skalarni registar):

$$f_3: s_j \times V_i \rightarrow V_k,$$

iii) instrukcija vektor-memorija (M označava radnu memoriju):

$f_4: M \rightarrow V_j$ za operaciju *load*

$f_5: V_i \rightarrow M$ za operaciju *store,*


vektorski registar v_i
vektorski registar v_j
vektorski registar v_k
protočna funkcijska jedinica
a)
vektorski registar v_i
skalarni registar s_j
vektorski registar v_k
protočna funkcijska jedinica
b)

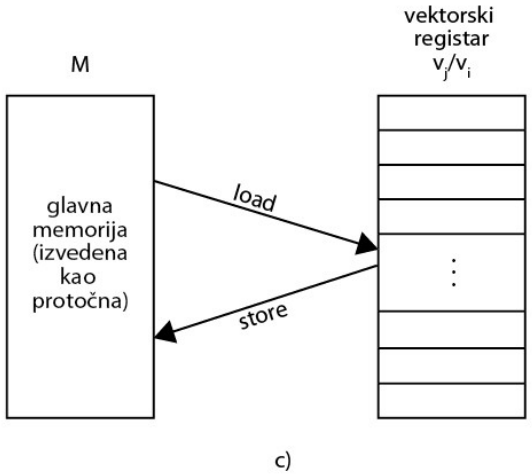
iv) instrukcije redukcije pretvaraju vektore u skalare.

Formalno se taj tip instrukcije može opisati kao:

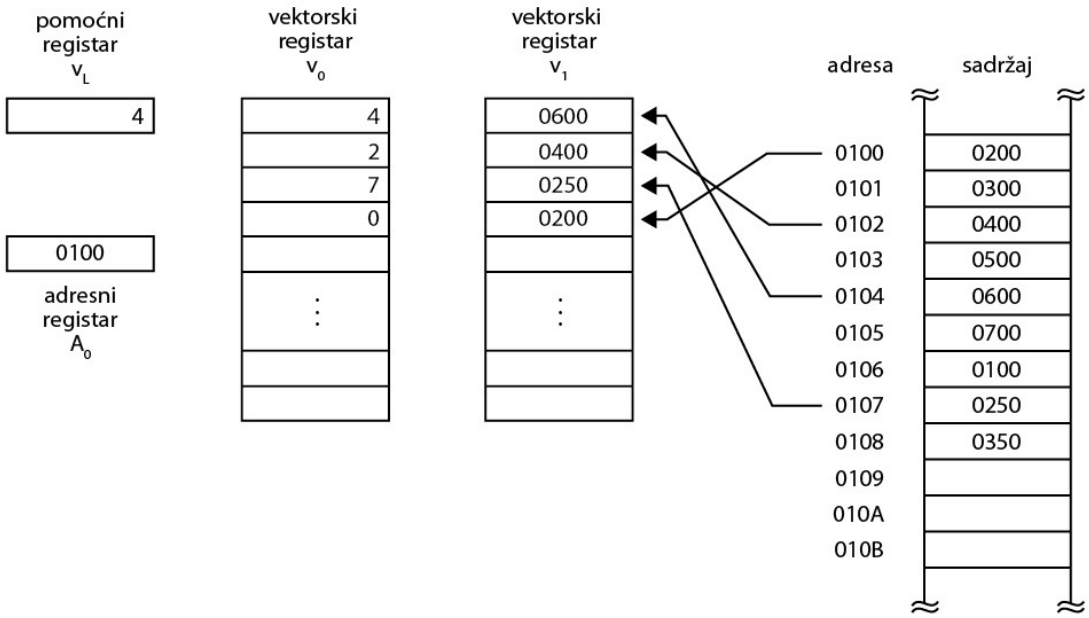
$$f_6: V_i \rightarrow s_j$$

$$f_7: V_i \times V_j \rightarrow s_j.$$

(npr. dohvat maksimalnog elementa (f_6); skalarni produkt (f_7))



c)



d) Okupljanje

v) instrukcije okupljanja (engl. *gather*) ili raspršivanja (engl. *scatter*)

$f_8: M \rightarrow V_1 \times V_0$ okupljanje

$f_9: V_1 \times V_0 \rightarrow M$ raspršivanje.

Operacijom okupljanja iz memorije se dohvaćaju elementi različiti od nule, i to tako da vektorski registar V_0 sadržava indekse (kazaljke) na podatke u memoriji, a vektorski registar V_1 sadržava podatke koji se iz memorije dohvaćaju i oblikuju tzv. rijetko popunjeni vektor.

Raspršivanje je obrnuta operacija u odnosu na okupljanje:
njome možemo u memoriju pohraniti rijetko popunjeni vektor.

vi) instrukcije maskiranja – to su instrukcije vektor-vektor koje dodatno zadaju i vektor maske V_m (engl. *mask vector*) koji određuje na kojim elementima će se operacija provesti.

Formalno, maskiranje može se opisati kao preslikavanje:

$$f_{10}: \; V_0 \times V_m \rightarrow V_1.$$

Npr. u vektorskom registru V_1 pohranjuju se elementi registra V_0 koji su različiti od 0 i za koje je odgovarajući element V_m različit od nule.

Primjer

Ilustrirajmo izvođenje operacije **y** = *s***x** + **y**
- **y** i **x** su 64 komponentni vektori dvostruke točnosti (64 bit),
- *s* je skalar dvostruke točnosti

Analizu ćemo provesti na skalarnom i vektorskom procesoru.

Pretpostavljamo da su na početku izvođenja **y**, **x** i s pohranjeni u memoriji:
- početna adresa **x** je na lokaciji $s0,
- početna adresa **y** je na lokaciji $s1,
- skalar s je na lokaciji $sp.

Programski odsječak za skalarni procesor izgleda ovako:

```
        ld      $f0, $sp        ; dohvati skalar s i pohrani ga
                                ; u floating-point registar f0
        addi    $r4, $s0,#512   ; gornja adresa lokacije na kojoj
                                ; se nalazi vektor x
opet:   ld      $f2, 0($s0)     ; dohvati x(i)
        mul     $f2, $f0,$f2    ; s × x(i)
        ld      $f4, 0($s1)     ; dohvati y(i)
        add     $f4, $f2, $f4   ;  s × x(i) + y(i)
        st      $f4, 0($s1)     ; pohrani rezultat na y(i)
        addi    $s0, $s0, #8    ; uvećaj indeks za x
        addi    $s1, $s1, #8    ; uvećaj indeks za y
        sub     $t0, $r4, $s0   ; razlika tekuće i krajnje adrese
        bne     $t0, $zero, opet ; petljaj dok razlika ne postane =0
```

(Opaska: adresna zrnatost memorije je bajtna, zato je gornja granica 512, tj. 64×8, gdje je 64 broj komponenti vektora, a svaka je njegova komponenta duljine 8 bajtova.)

Programski odsječak za vektorski procesor izgleda ovako:

```
ld      $f0, $sp        ; dohvati skalar s i pohrani ga u
                        ; floating-point registar f0
ldv     $v1, 0($s0)     ; dohvati vektor x i pohrani ga u
                        ; vektorski registar v1
mulvs   $v2, $v1,$f0    ; množenje vektora x sa
                        ; skalarom s
ldv     $v3, 0($s1)     ; dohvati vektor y i pohrani ga u
                        ; vektorski registar v3
addv    $v4, $v2,$v3    ; pribroji y produktu sx
stv     $v4, 0($s1)     ; pohrani rezultat
```

Usporedba gornjih programskih odsječaka:

- vektorski program značajno reducira promet instrukcija – on zahtijeva samo šest instrukcija, dok se kod skalarnog procesora zahtijeva skoro 600 instrukcija (programska petlja).
- druga zanimljiva razlika je u učestalosti hazarda – za skalarni procesor svaka *add* instrukcija mora čekati na *mul* instrukciju te svaka *st* instrukcija mora čekati na *add* instrukciju.
- za vektorski će procesor svaka vektorska instrukcija biti u zastoju samo za prvi element svakog od vektora, tako da će ostali elementi glatko protjecati kroz protočnu strukturu.

Vektorske instrukcije prvo su korištene u tzv. superračunalima (npr. Cray 1, 1976)

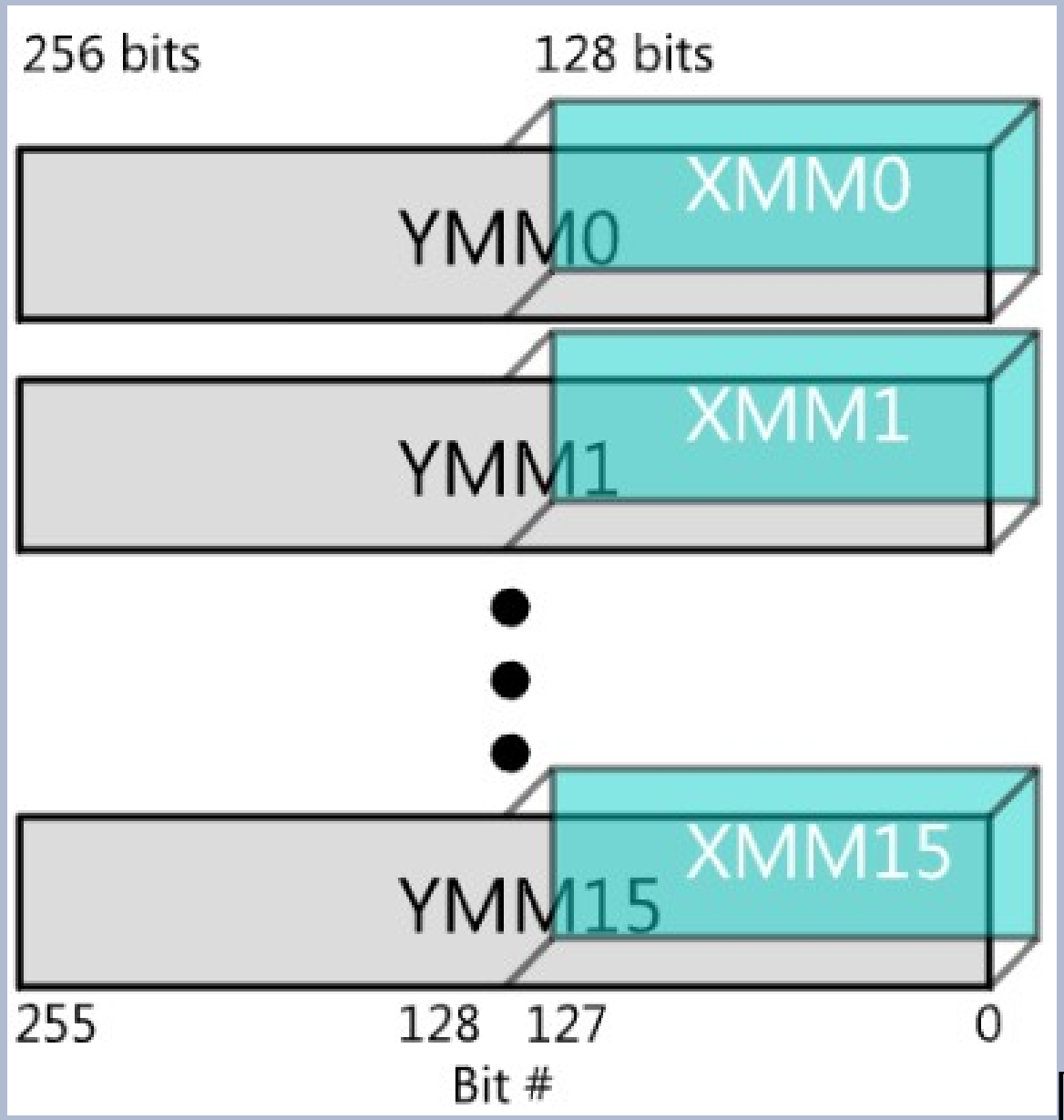
U novije vrijeme, koriste se i u računalima opće namjene (x86: MMX, SSE, AVX, ARM: Neon)

U nastavku ćemo ukratko predstaviti vektorska proširenja za arhitekturu **x86** (MMX, SSE, AVX)

Primjeri će koristiti Intelovu sintaksu (na gcc-u treba zadati `.intel_syntax noprefix`)

Vektorski registri ekstenzija arhitekture x86

- xmm (SSE): 16×128 bit = 16×4×fp32,
- ymm (AVX): 16×256 bit = 16×8×fp32,
- zmm (AVX-512): 32×512 bit = 32×16×fp32
- donja polovica od ymm je xmm; jednako za zmm i ymm.


256 bits
128 bits
YMM0
XMM0
YMM1
XMM1
YMM15
XMM15
255
128
127
0
Bit #

[lomont11]

Vektorske instrukcije (SSE):

Data transfer	Arithmetic	Compare
MOV{A/U}{SS/PS/SD/PD} xmm, mem/xmm	ADD{SS/PS/SD/PD} xmm,mem/xmm	CMP{SS/PS/SD/PD}
	SUB{SS/PS/SD/PD} xmm,mem/xmm	
MOV {H/L} {PS/PD} xmm, mem/xmm	MUL{SS/PS/SD/PD} xmm,mem/xmm	
	DIV{SS/PS/SD/PD} xmm,mem/xmm	
	SQRT{SS/PS/SD/PD} mem/xmm	
	MAX {SS/PS/SD/PD} mem/xmm	
	MIN{SS/PS/SD/PD} mem/xmm	

[patterson13]

A/U: aligned/**unaligned**
S/P: scalar/**packed**
S/D: **single precision**/double precision
H/L: high half/low half

Primjeri

Zbroji vektore na koje pokazuju `rsi` i `rdx` te upiši rezultat na lokaciju na koju pokazuje `rdi`:

```
movups xmm1, WORD PTR [rsi]
movups xmm2, WORD PTR [rdx]
addps xmm1, xmm1, xmm2
movups WORD PTR [rdi], xmm1
```

Pomnoži `ymm1` i `ymm3` po elementima, dodaj `ymm2` i spremi natrag u `ymm3` (fused multiply-add):

```
fmadd ymm1, ymm2, ymm3
```

Postavi memorijski operand u sve elemente vektorskog registra (vector boradcast):

```
vbroadcastss ymm1, WORD PTR [rax+r8]
```