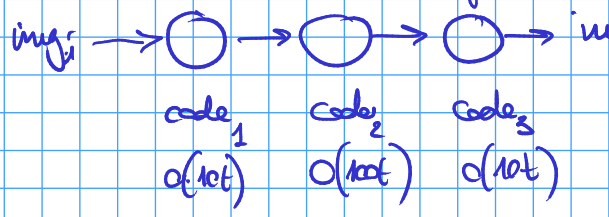
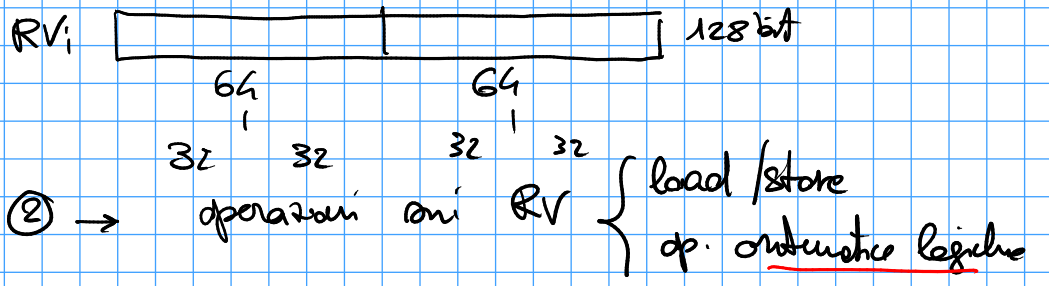
VETTORIZZAZIONE

→ Sol × ridurre "frazioni seriali"



Hw vettoriale — INTEL SSE, SSEx, AVX, PAI 512 bit
POWER PC Altivec

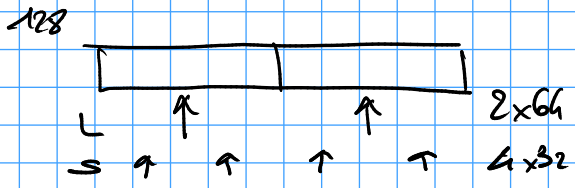
① → Registri vettoriali (unità)
(reg. grandi, dim. grandi, unione di reg + piccoli)



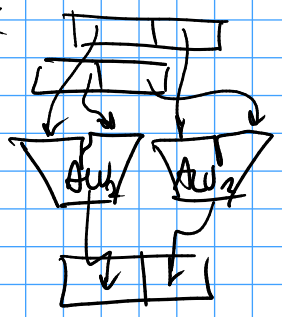
② → operazioni on RV { load/store
op. aritmetiche logiche

LOAD Rbase, Rindice, Rtarget — M[Rbase + Rindice] → Rtarget
STORE " " Rsource ←
ADD Ri, Rj, Rk — Ri + Rj → Rk
SUB
MUL
DIV

LOADV Rbase, Rindice, RVtarget
STOREV
ADD V$^{L}_{S}$ RVi, RVj, RVk



ADDV




```
for(i=0 ; i<N ; i++)
  c[i] = a[i] * b[i];
```

Non vect
LD "ai"
LD "bi"
MUL
ST

vect
LOADV "ai, ai+1, ..., ai+3"
LOADV "bi ... bi+3"
MULV
STOREV

soluzione preferibile

Compilatore vettorizzante

(gcc / g++ , icc / icpc)

vettorizzazione

≈ programma

"ASM"

Non ci piace

(CARATTERISTICHE)

loop × parallelizzabile

- determinati
- 1 punto di ingresso 1 punto di uscita

```
for(int i=0; i<N; i++) {
    if(a[i]==0) break;
}
```

- "codice lineare"

non avere branch nel codice

```
for ( ) {
    switch(a[i])
        case 0 : a[i]++
        case -1 : a[i]=0
        default : a[i]=
    }
```

a[0]	a[1]	a[2]	a[3]

eccezione

if then else (mascherato)

if(c) then(S_1) else(S_2)

- parallelizzazione cicli interni

```
for(i= ...
{ for(j= ...
    S1
```

- no chiamate di funzione

```
for(i= ...
    a[i] = f(b[i]);
```

eccezione → f inline

DIPENDENZE (LAYOUT di MEMORIA)

x dipende da y $\begin{matrix} x \\ y \end{matrix} \in [read, write]$

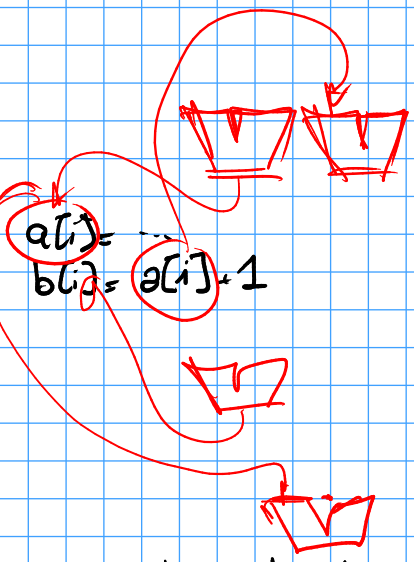
read AFTER read (don't care)
c[i] = a[i]*2
d[i] = a[i]/4

write AFTER write
c[i] = ...
c[i] = ...

read AFTER write (dipendenza)
write AFTER read (anti-dipendenza)
dip di output

a[i] = ...
b[i] = a[i] + 1

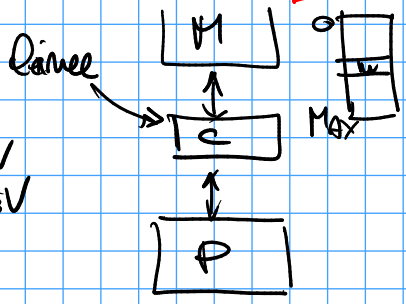
a[i] = b[i] + 1
b[i] = ...

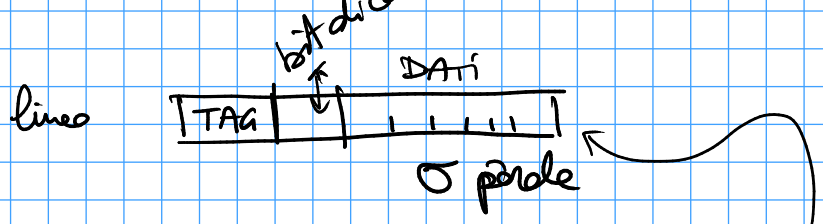


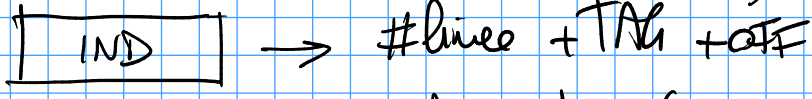
Layout di Memoria

allineamento dei dati

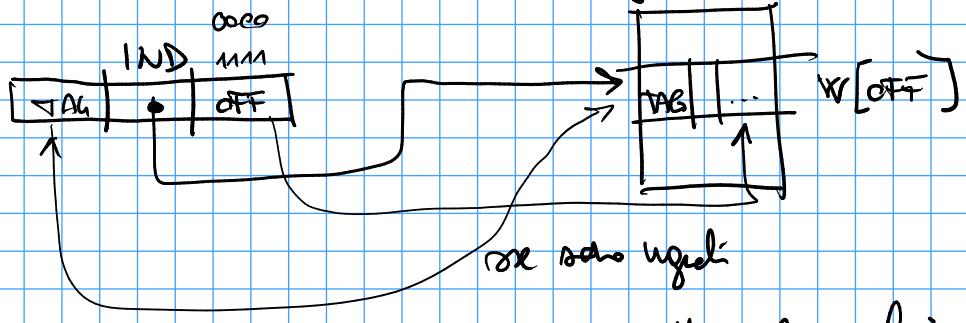
LOADV
STOREV



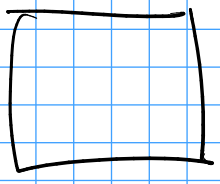




Cache indirizzamento diretto (codice)



σ = #parole x linee

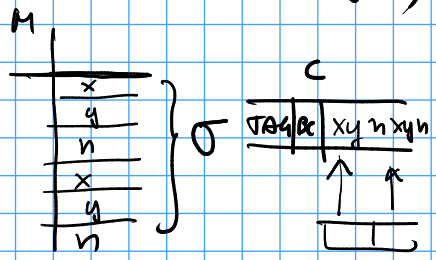


LOADV STOREV

Memory layout ⇒ dati "allineati" (ML1)

```
struct { float x;
         float y;
         int n; } [ ]
```

"x[i] = n[i] . y[i];"




```
struct { } vec [N];
for (i=0 ... ) {
    vec[i] . x = vec[i] . n * vec[i] . y;
}
```

(Load)

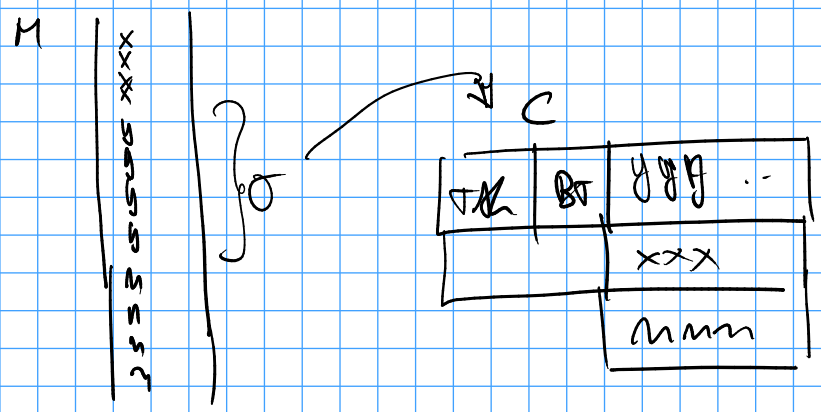
accessi non contigui (ML2)

⇓

resi contigui

vector of struct → struct of vector

```
struct { float x[N]; float y[N]; int n[N]; } a;
for (i= .... ) {
    a.x[i] = a.y[i] * a.n[i];
```



gcc g++ -O2 -O3

-ftree-vectorizer-verbose= 0 1 → loop not 2 → vect 3

__declspec(align(16)) int x[128];

restrict (var) ptr

restricted (x)
(float *x, float *y)
int n

for (i=0; i<n; i++)
x(i) = y(i);