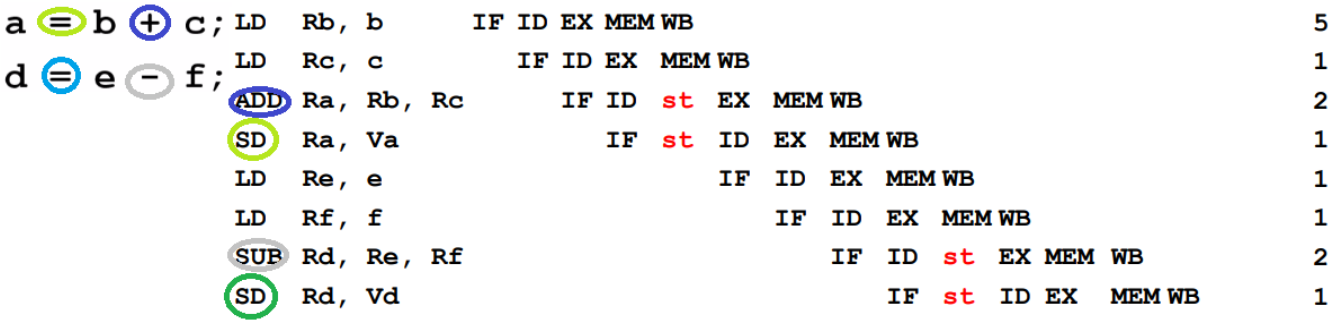
Ci sono due principali approcci:

- **Dinamico:** dipende dall'hardware. Principalmente usato per ambienti server/desktop
- **Statico:** dipende dal software(ad esempio il compilatore). Principalmente usato per ambienti embedded.
  E' possibile anche usare un approccio ibrido.

Rescheduling

Si basa sul **riordinare** delle istruzioni **indipendenti** tra di loro in un *basic block*, ovvero un blocco di istruzioni senza branch tranne che all'inizio o alla fine.

Esempio



Ci sono varie dipendenze che generano stalli.
Una soluzione è quella di riordinare le istruzioni:

```
LD   Rb, b       IF ID EX MEM WB
LD   Rc, c          IF ID EX  MEM WB
LD   Re, e             IF ID  EX  MEM WB
ADD  Ra, Rb, Rc           IF  ID  EX  MEM WB
LD   Rf, f                   IF  ID  EX  MEM WB
SD   Ra, Va                     IF  ID  EX  MEM WB
SUB  Rd, Re, Rf                    IF  ID  EX  MEM WB
SD   Rd, Vd                            IF  ID  EX  MEM  WB
```

Loop-level parallelism

Ogni iterazione di un loop è indipendente dalle altre, quindi possono essere interfogliate.
Ciò può essere sfruttato per il **loop-level parallelism** attraverso:

- **loop unrolling** (statico o dinamico)

- **SIMD**

Loop unrolling

Si basa sul replicare il corpo del loop **N** volte.
Ciò riduce il numero di *branch* tuttavia la vera potenza sta nel permettere un **rescheduling** migliore, dal momento che la dimensione del basic block aumenta.
Ovviamente aumenta anche la dimensione del codice.

```
for (i=0;i<N;i++ )
{
    body
}
```

```
for (i=0;i<N/4;i++ )
{
    body
    body
    body
    body
}
```

SIMD

Sta per **Single Instruction** stream **Multiple Data** stream.
Con dei processori particolari(GPU o vettoriali) è possibile effettuare delle istruzioni in parallelo su vettori di elementi.

Dipendenze

Per sfruttare **ILP** bisogna dunque riconoscere le varie dipendenze tra le istruzioni.
Le dipendenze fanno parte del programma.
Gli hazard sono proprietà dell'organizzazione della pipeline.
Esistono 3 tipi di dipendenze.

Data dependence

Un'istruzione *i* è data dependent da una istruzione *j* se:

- l'istruzione *i* produce un risultato usato dall'istruzione *j*
  oppure
- l'istruzione *j* è dipendente sull'istruzione *k* e l'istruzione *k* è dipendente dall'istruzione *i*.

Memory dependencies

Scovare dipendenze legate agli accessi alla memoria è molto difficile, dal momento che accessi diversi alla stessa cella possono sembrare molto diversi.
Solo a run-time è possibile scovarle, oppure usando un approccio statico, supponendo che

Name dependence

Accade quando due istruzioni si riferiscono allo stesso registro.
Due tipi:

- **antidipendenza:** quando una istruzione usa un determinato registro e l'istruzione successiva ci scrive sopra.Non è dunque una proprio dipendenza ma bisogna tenerne conto durante il rescheduling.
- **output dipendenza:** quando due istruzioni scrivono nello stesso registro/locazione di memoria

Register Renaming

Per risolvere queste **name dependencies**, basta rinominare i registri che causano le dipendenze.

Correlazione Hazard e Dependecies

- **Read After Write - RAW:** Corrispondono ad una **true data dependecy**.
- **Write After Write - WAW:** Nascono da **output dipendenze**
- **Write After Read - WAR:** Nascono dalle **antidipendenze**

Control dependencies

Esiste quando un'istruzione dipende da da un **branch**.