Si basa sul riordinamento, fatto a livello hardware, delle istruzioni in modo da ridurre gli statti della pipeline
Permette di **identificare** delle dipendenze sconosciute a compile time, **semplifica** il lavoro del compilatore e permette di far girare lo stesso codice su **diversi processori pipelined**

Si basa sul cambiare l'esecuzione da **in-order** a **out-of-order**, tuttavia ciò introduce,come abbiamo visto con la pipeline FP, degli **hazards**, come **WAR** e **WAW**

Per supportare ciò, si *splitta* in due lo stadio di **Decode:**

- **issue:** si occupa di decodificare le istruzioni e di controllare gli hazard strutturali
- **read operands:** aspetta finchè non ci siano data hazard e poi legge operandi

Stadio di ID Issue

Legge le istruzioni da un registro/coda(scritto dallo stadio IF) in **ordine.**
Le istruzioni vengono mandate allo stadio di read **operands** in caso non ci siano dipendenze di nessun tipo.

Approccio di Tomasulo

Si basa su:

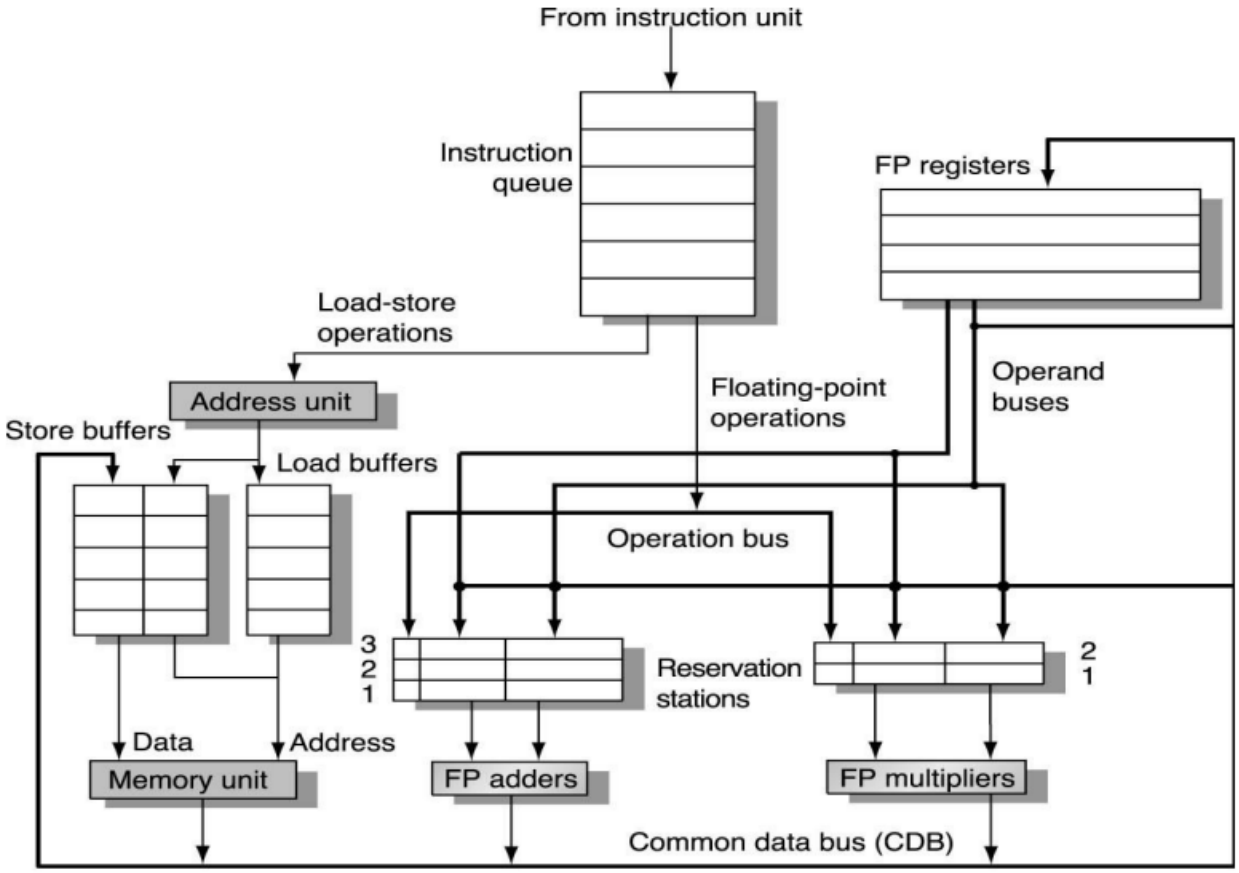
- tracciare la **disponibilità** degli operandi
- introdurre un concetto di *virtual* register **renaming**

Reservation stations

Ulteriori registri che bufferano le operazioni in attesa di essere completate.
In modo univoco identificano le istruzioni all'interno della pipeline.

Common Data Bus (CDB)

I risultati vengono passati direttamente ad altre *unità funzionali* piuttosto che passare attraverso dei registri.
Dunque tutti i risultati delle **unità funzionali** passano attraverso un **Common Data Bus** il quale **raggiunge tutti** i punti dell'architettura(tranne il load buffer).
Ciò permette a tutte le unità che aspettano un operando di caricarlo simultaneamente quando è disponibile.



Questa è la nuova struttura dello stadio ID.
Le istruzioni vengono inserite in una coda(**FIFO**).
Uscite dalla coda, vanno in delle **reservation stations**(sale d'attesa), dove aspettano che gli operandi siano disponibili.
Gli operandi possono venir dal **CDB** o dal **register file**.

Esecuzione delle istruzioni

Avviene in tre fasi.

Issue

1. Viene presa l'istruzione dalla coda(**FIFO**).
2. Se nessuna **reservation station** è disponibile accade un **hazard strutturale** e l'istruzione *stalla* finchè non se ne libera una
3. Se c'è una **reservation station** disponibile, l'istruzione viene inoltrata lì.
    1. Se gli operandi sono disponibili, vengono inoltrati alla **reservation station**
    2. Se non lo sono, le **unità funzionali** responsabili per la loro generazione vengono registrati

Execution(istruzioni normali)

Quando un operando appare nel **CDB**, viene letto dalla **reservation unit**.
Appena tutti gli operandi diventano disponibili nella **reservation unit**, l'istruzione viene eseguita.
In questo modo si evitano i **RAW** hazards

Execution(load/store)

1. Appena il *base* register è disponibile, viene calcolato l'effective address e scritto nel **buffer load/store**.
2. **Load:** appena la memoria è disponibile l'istruzione viene eseguita
3. **Store:** aspetta per l'operando che viene scritto in memoria. Appena la memoria è disponibile l'istruzione viene eseguita

Execution(branches)

Per mantenere la corretta gestione delle eccezioni, nessuna istruzione viene eseguita finchè tutti i branch precedenti l'istruzione non sono stati completati.
La **speculazione** può migliorare ciò

Write result

Quando il risultato di un'istruzione è disponibile, viene scritto **subito** sul **CDB** e da lì nei registri e nelle **unità funzionali** che li aspettano.
Le **store** scrivono i risultati in memoria in questo step

Instruction identifiers

Ogni **reservation station** viene associato ad un identificatore univoco.
Questo identificatore identifica anche l'operando che l'istruzione produce

Op	V_j	V_k	Q_j	Q_k	A	Busy

- **Op:** tipo dell'operazione(ADD,MUL etc.)
- V_j,V_k**:** i valori **numerici** degli operandi(10,20,120 etc), può essere anche vuoto in caso i valori non siano ancora disponibili
- Q_j,Q_k**:** identificatori delle istruzioni che producono questi operandi
  - Supponiamo di avere ADD R3,R1,R2 dove R1 dipende da MUL R1,R5,R6.
  Nel campo Q_j ci sarà MUL1, visto che R1 dipende dall'istruzione *MUL1*
- **A:** Usato solo da **load/store**. Si salva prima l'immediate field, e poi l'effective address
- **Busy:** Se la **reservation station** sta aspettando degli operandi o meno(flag da 1 bit)

Register File

Ogni entry nel *register file* contiene un campo Q_i che indica il numero di reservation station che contiene istruzioni il cui risultato deve essere salvato in tale registro.
Se è 0/null significa che nessuna istruzioni attiva scrive su tale registro

Register status

Contiene per ogni registro l'ultima istruzione che lo usa

Loop handling

Grazie all'architettura di Tomasulo, il loop unrolling non è necessario

Dynamic disambiguation

L'esecuzione out of order di **load/store** può comunque causare **RAW/WAR** hazards. Quindi per ogni **load/store** bisogna effettuare dei controlli aggiuntivi