Lezione 2 – Problemi di esecuzione di programmi con CPU *pipeline*

Architettura degli elaboratori

Modulo 5 - Principali linee di evoluzione architetturale

Unità didattica 2 - Strutture *pipeline*

Nello Scarabottolo

Università degli Studi di Milano - Ssri - CDL ONLINE

Dipendenze (*dependencies*)

Durante l'esecuzione di programmi da parte di una CPU *pipeline*, si incontrano situazioni che comportano il rallentamento delle attività, quindi la perdita di efficacia.

- ***Control* dependency**:

  il flusso di esecuzione delle istruzioni interrompe il normale comportamento sequenziale.

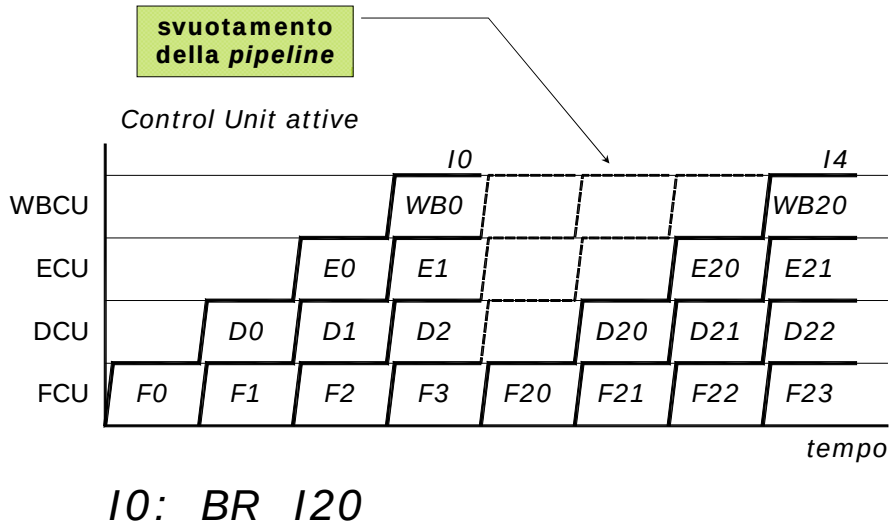
- ***Data* dependency**:

  due istruzioni vicine utilizzano uno stesso dato.

- ***Resource* dependency**:

  due istruzioni entrano in conflitto per una risorsa del sistema.

Control dependency



I0: BR I20

Branch prediction

I salti sono comunque una minoranza delle istruzioni (10% ÷ 20%).

I salti incondizionati possono essere risolti a livello di FCU.

Per migliorare l'efficienza con i salti condizionati, approccio statistico:

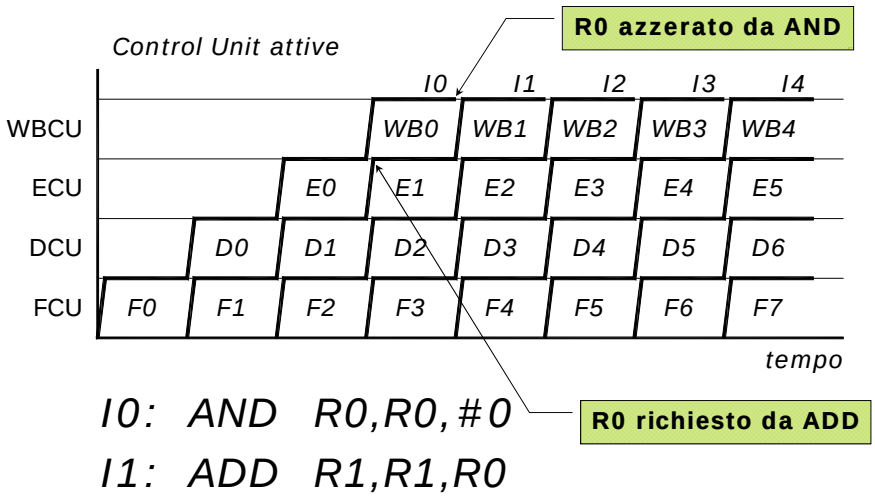
- se ho già incontrato l'istruzione di salto, mi comporto come la volta precedente;
- se NON ho mai incontrato l'istruzione di salto:
  - se è un salto all'indietro, provo a saltare (**cicli**);
  - se è un salto in avanti, provo a NON saltare (costrutti **if...then...else...**).

Branch Prediction Table (*BPT*)

Branch address	Branch destination	Statistics
0	20	T

- Quando FCU incontra istruzione di salto, consulta **Branch address**;
- se aveva già incontrato l'istruzione, si adegua a **Statistics** (**T**aken / **N**ot taken) e eventualmente salta a **Branch destination**;
- se non aveva incontrato l'istruzione, inserisce in BPT e salta indietro / NON salta in avanti;
- quando ECU risolve il salto, aggiorna BPT e in caso di scelta "sbagliata" di FCU provoca svuotamento della *pipeline*.

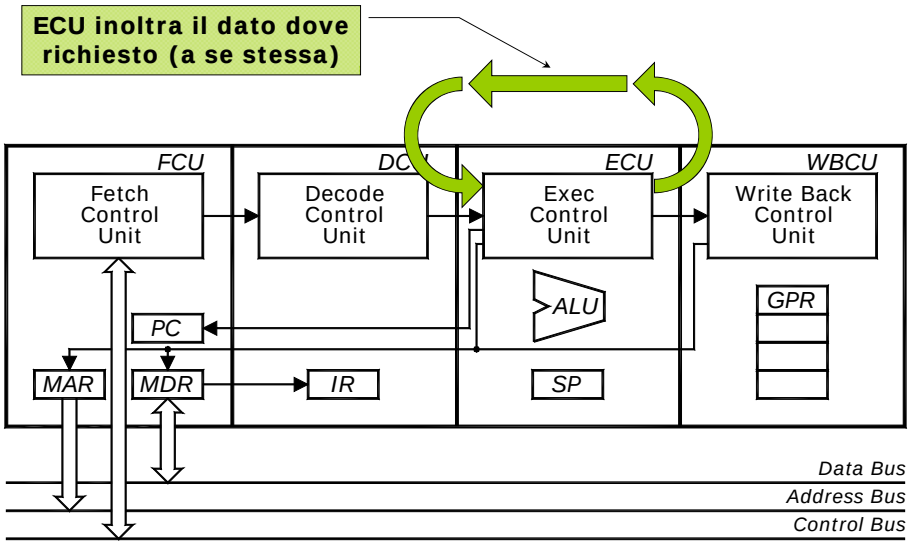
Data dependency



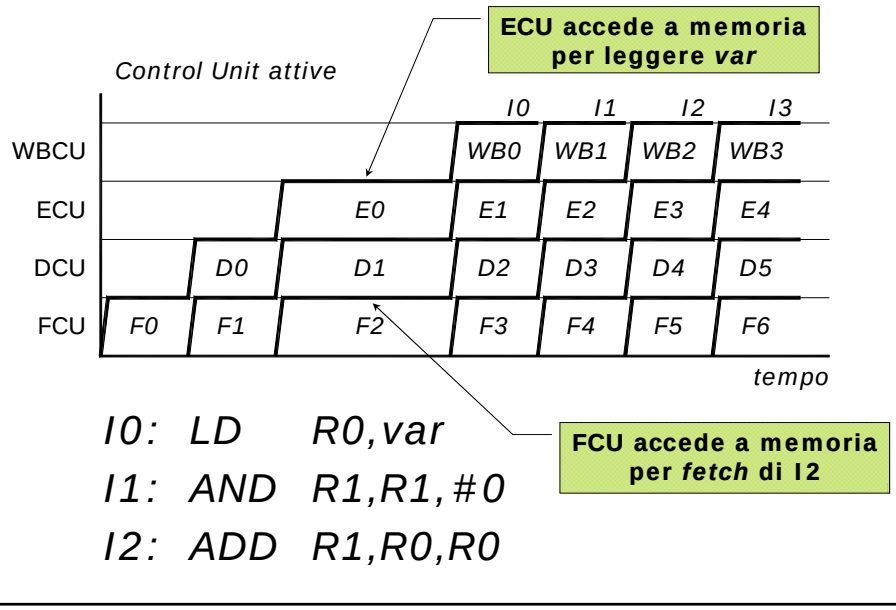
I0: AND R0,R0,#0

I1: ADD R1,R1,R0

Data forwarding

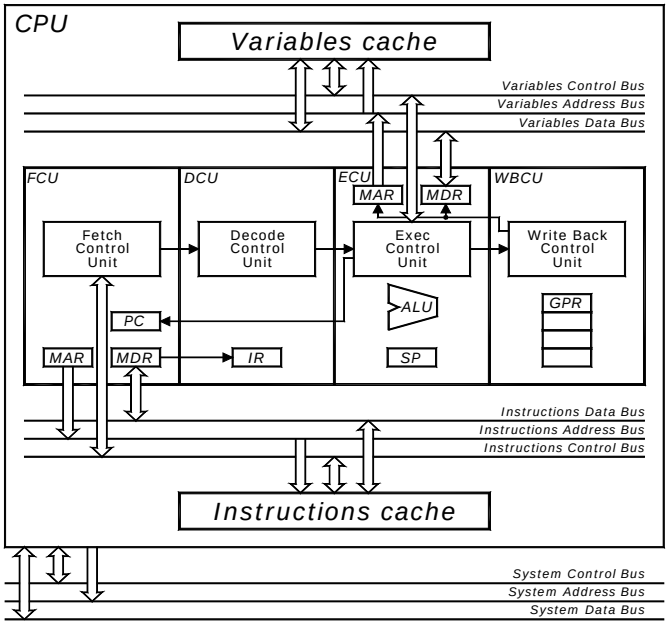


Resource dependency




```
I0:  LD    R0,var
I1:  AND   R1,R1,#0
I2:  ADD   R1,R0,R0
```

Macchina di *Harvard*



Macchina di *Harvard*

La macchina di Harvard ha spazi di indirizzamento separati per istruzioni e dati variabili.

Con due *cache* a bordo della CPU elimino i conflitti fra fasi di fetch e fasi di accesso ai dati variabili, senza duplicare il *System Bus*.

Rimane il conflitto fra ECU e WBCU in caso di letture e scritture simultanee di dati variabili:

- si ricorre a tecniche di compilazione ottimizzata, che riordinano le istruzioni macchina in modo che una istruzione di STORE e una di LOAD non entrino in conflitto sull'accesso al ***Variables Bus***.

In sintesi...

I legami fra le istruzioni in esecuzione e i loro conflitti di accesso alle risorse possono provocare rallentamenti della *pipeline* dovuti a:

- *control dependencies*;
- *data dependencies*;
- *resource dependencies*.

Si riesce a ridurre le inefficienze ricorrendo a soluzioni architetturali anche complesse:

- *branch prediction*;
- *data forwarding*;
- CPU con *cache* di primo livello separata per istruzioni e variabili.

Chiusura

Fine della lezione

...e dell'insegnamento!