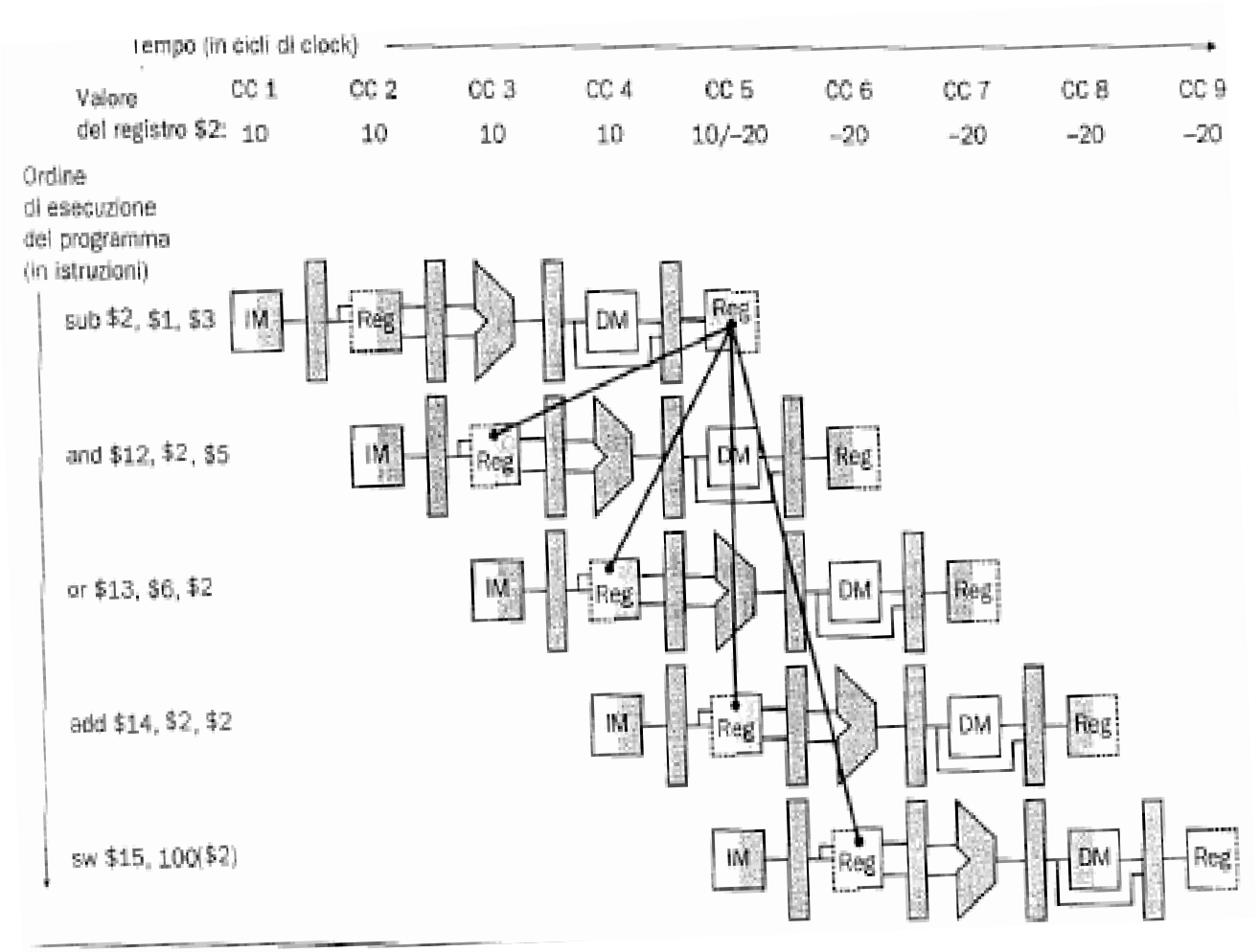
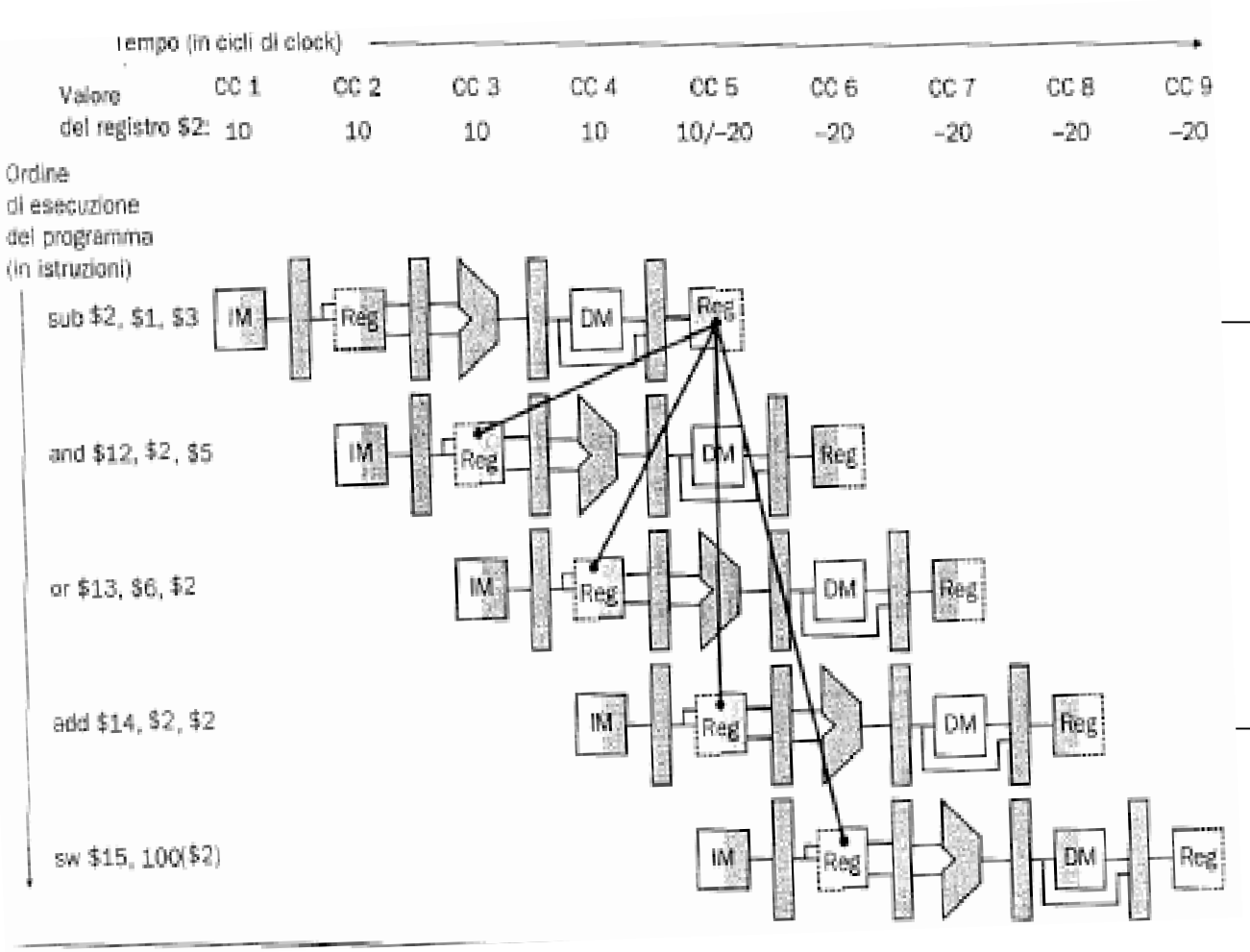
E’ possibile risolvere questi conflitti evitando di mettere la CPU in stallo?

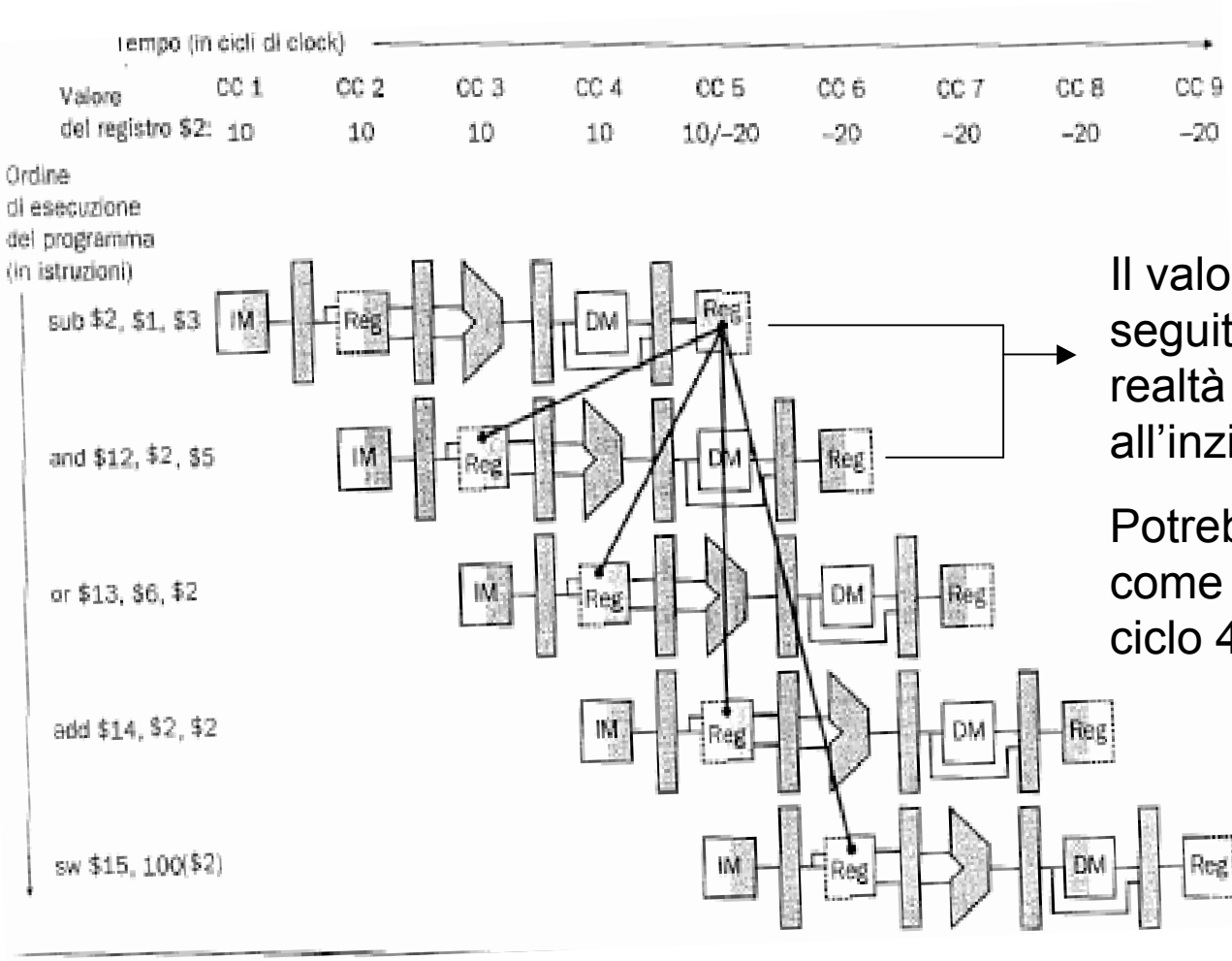


E’ possibile risolvere questi conflitti evitando di mettere la CPU in stallo?



- Scrittura nel register file nella prima metà del ciclo di clock / lettura nella seconda metà

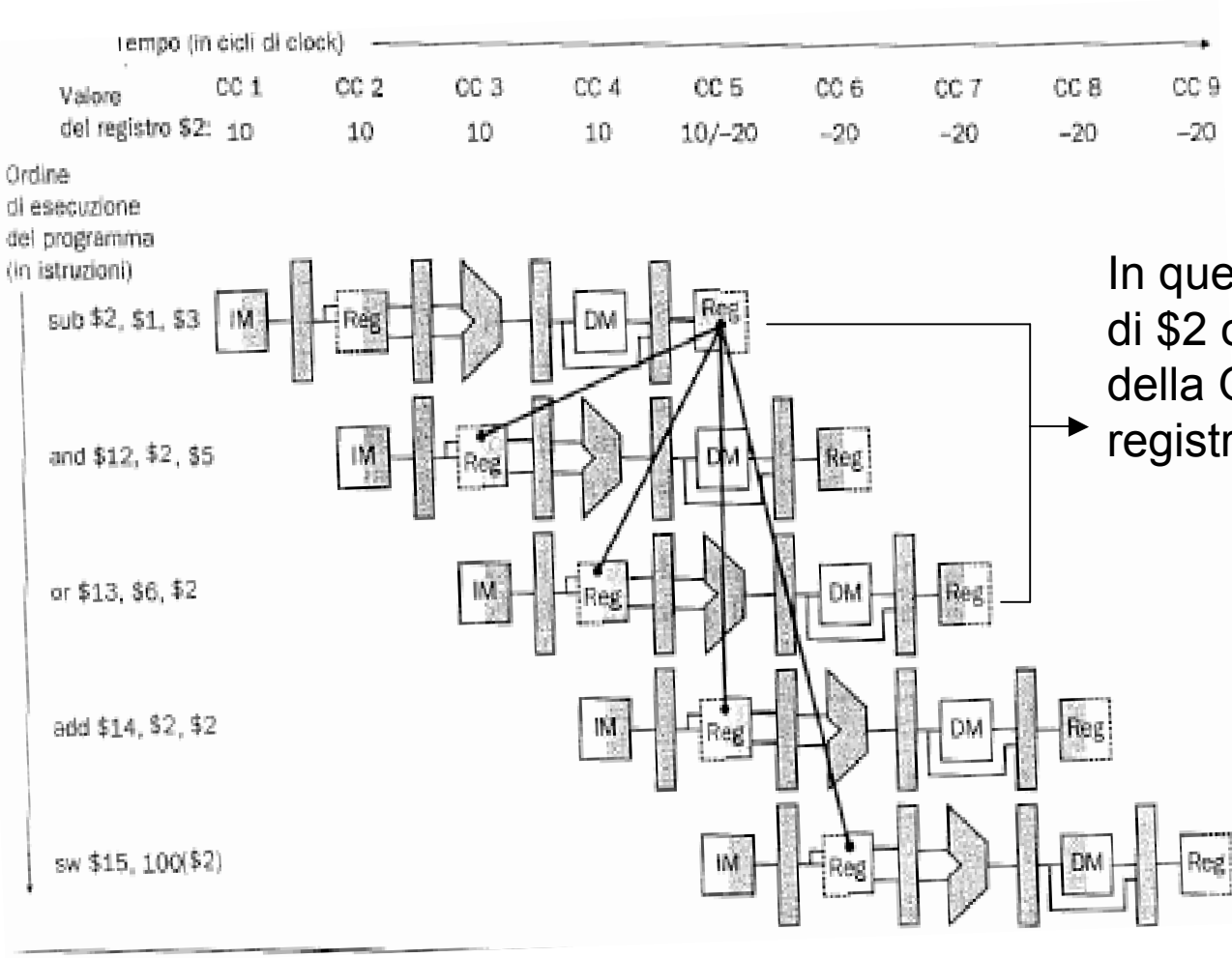
E’ possibile risolvere questi conflitti evitando di mettere la CPU in stallo?



Il valore aggiornato di $2 (in seguito all’esec. della SUB) è in realtà già disponibile in EX/MEM all’inzio del ciclo 4:

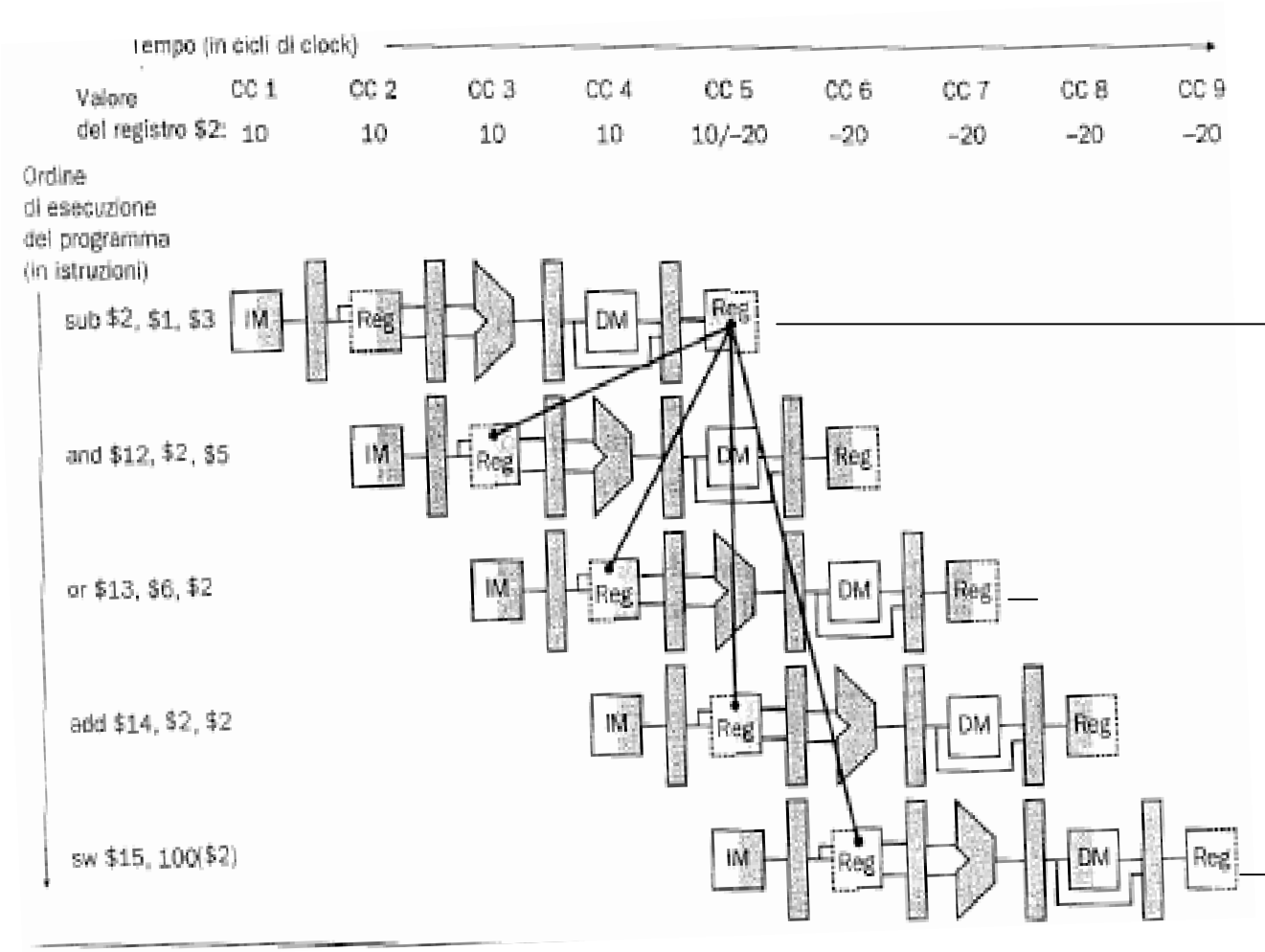
Potrebbe quindi essere usato come operando dalla ALU nel ciclo 4 per eseguire la AND

E’ possibile risolvere questi conflitti evitando di mettere la CPU in stallo?



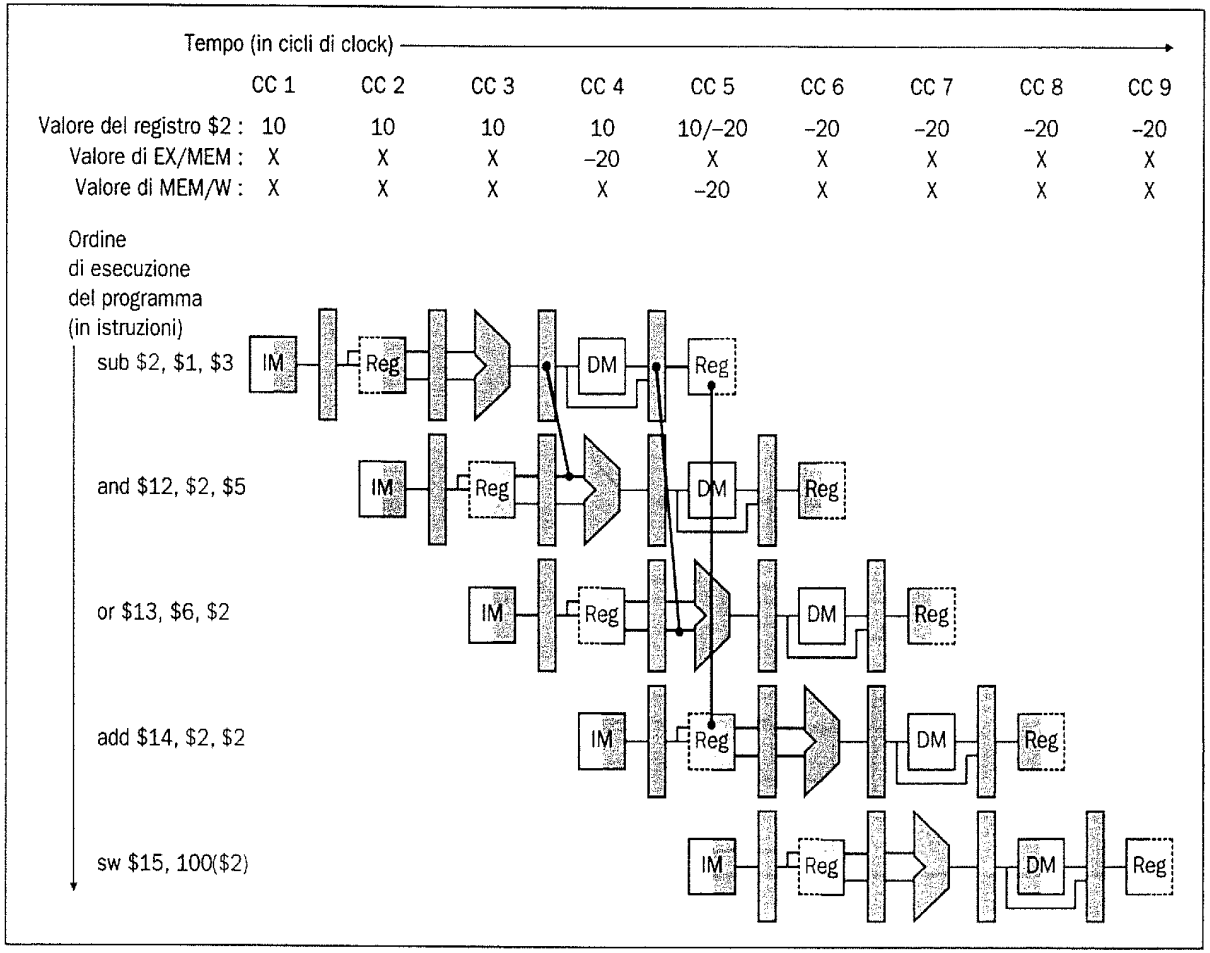
In questo caso il valore aggiornato di $2 occorre nel ciclo 5 (fase exec della OR), ed è disponibile nel registro di pipeline MEM/WB

E’ possibile risolvere questi conflitti evitando di mettere la CPU in stallo?



NON E’ UN CONFLITTO!

Propagazione da EX/MEM e MEM/WB agli ingressi della ALU



Unità Propagazione Dati: Condizioni di Abilitazione

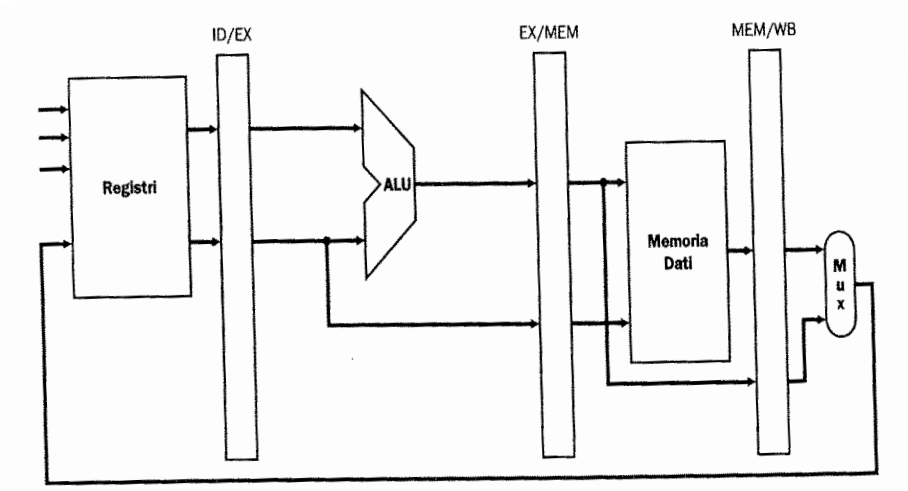
1) Istruzioni precedenti (ovvero negli stadi MEM o WB) intendono accedere in scrittura al register file:
   EX/MEM.RegWrite OR MEM/WB.RegWrite

2) Registro destinazione nelle fasi EX e MEM coincidente con uno degli operandi della ALU (ovvero ID/EX.RegistroRs e ID/EX.RegistroRt):
   EX/MEM.RegistroScrittura=ID/EX.RegistroRs
   EX/MEM.RegistroScrittura=ID/EX.RegistroRt
   MEM/WB.RegistroScrittura=ID/EX.RegistroRs
   MEM/WB.RegistroScrittura=ID/EX.RegistroRt

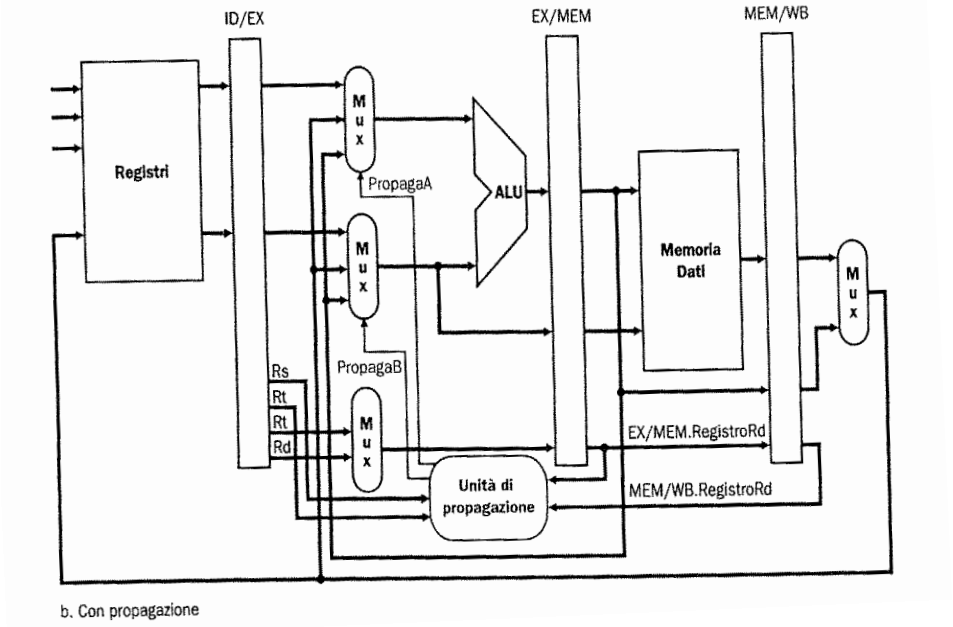
3) Dobbiamo trascurare accessi in scrittura su $0, poiché rimane in ogni caso immutato:
   EX/MEM.RegistroScrittura!=0
   MEM/WB.RegistroScrittura!=0

4) se un registro usato in EX viene scritto sia in MEM che in WB (ovvero dalle due istruzioni precedenti) occorre propagare dalla fase di MEM (ovvero il risultato dell'istruzione immediatamente precedente) che contiene il valore più recente:
   EX/MEM.RegWrite AND (EX/MEM.RegistroRd == ID/EX.RegistroRs) ⇒ non propagare Rs da MEM/WB
   EX/MEM.RegWrite AND (EX/MEM.RegistroRd == ID/EX.RegistroRt) ⇒ non propagare Rt da MEM/WB

Schema Circuitale Semplificato



a. Senza propagazione



b. Con propagazione

Controllo Multiplexer	Sorgente	Spiegazione
PropagaA=00	ID/EX	Il primo operando della ALU proviene dal register file.
PropagaA=10	EX/MEM	Il primo operando della ALU è propagato a partire dal precedente risultato della ALU.
PropagaA=01	MEM/WB	Il primo operando della ALU è propagato dalla memoria dati o da un precedente risultato della ALU.
PropagaB=00	ID/EX	Il secondo operando della ALU proviene dal register file.
PropagaB=10	EX/MEM	Il secondo operando della ALU è propagato a partire dal precedente risultato della ALU.
PropagaB=01	MEM/WB	Il secondo operando della ALU è propagato dalla memoria dati o da un precedente risultato della ALU.

1. criticità EX:

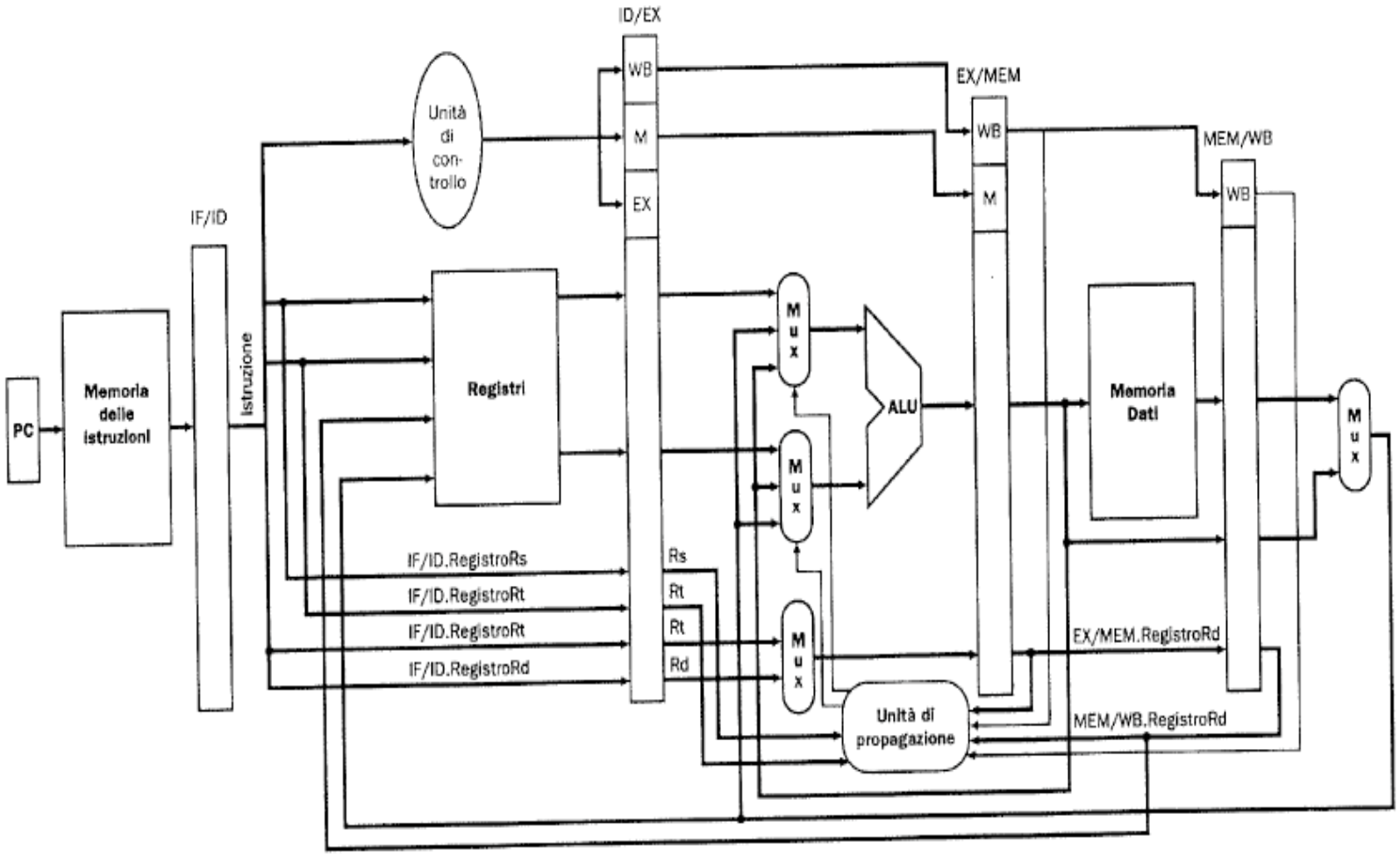
se (EX/MEM.RegWrite
e (EX/MEM.RegistroRd ≠ 0)
e (EX/MEM.RegistroRd = ID/EX.RegistroRs)) PropagaA = 10

se (EX/MEM.RegWrite
e (EX/MEM.RegistroRd ≠ 0)
e (EX/MEM.RegistroRd = ID/EX.RegistroRt)) PropagaB = 10

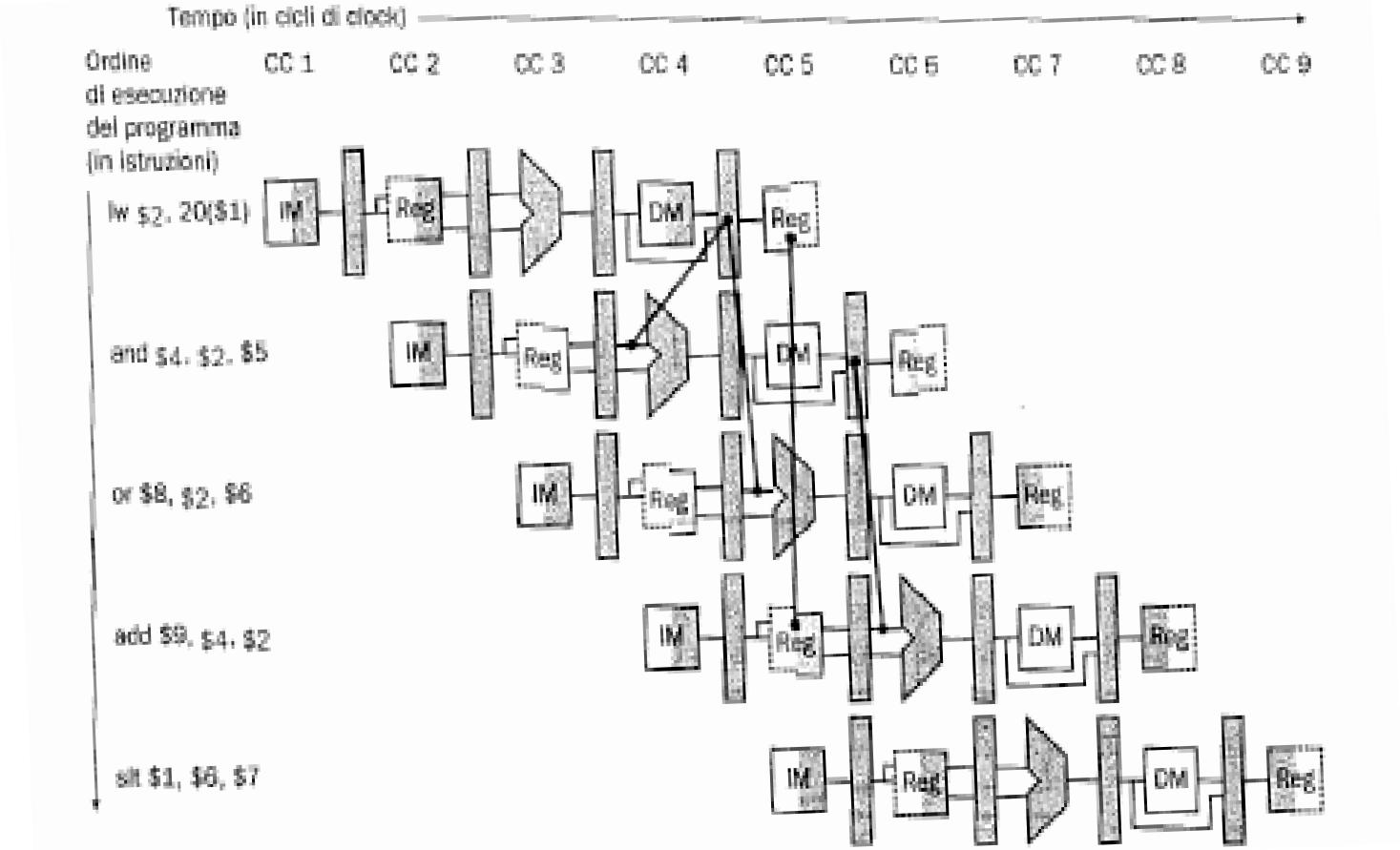
2. Criticità MEM:
se (MEM/WB.RegWrite
e (MEM/WB.RegistroRd!=0)
e(EX/MEM.RegWrite e EX/MEM.RegistroRd!=ID/EX.RegistroRs)
e (MEM/WB.RegistroRd=ID/EX.RegistroRs)) PropagaA=01

se (MEM/WB.RegWrite
e (MEM/WB.RegistroRd!=0)
e(EX/MEM.RegWrite e EX/MEM.RegistroRd!=ID/EX.RegistroRt)
e (MEM/WB.RegistroRd=ID/EX.RegistroRt)) PropagaA=01

Schema Circuitale Completo
(escludendo salti ed estensione segno)



La propagazione non è sempre sufficiente



Il valore aggiornato di $2 serve per la AND all'inizio ciclo 4, ma è disponibile solo alla fine del ciclo 4…. in questo caso occorre introdurre un ciclo di stallo