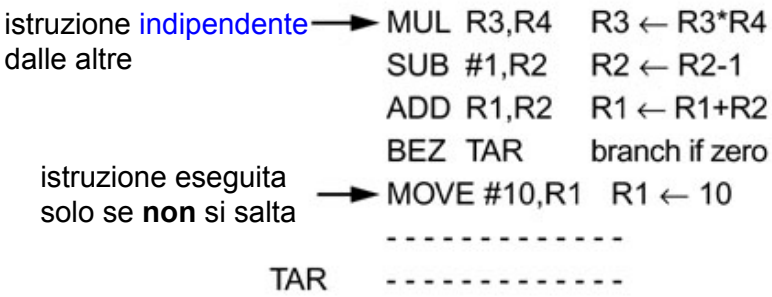
Soluzioni per salti condizionati

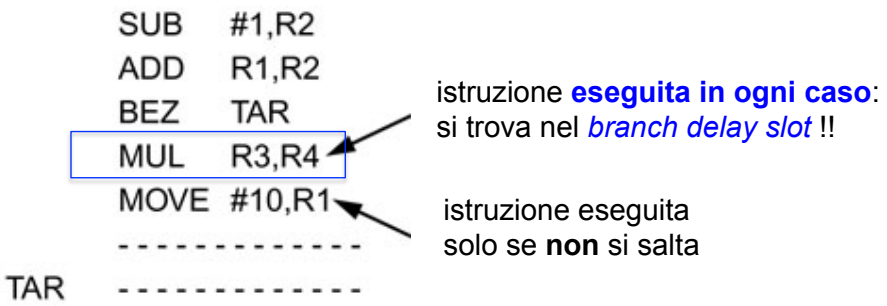
1. flussi multipli (*multiple streams*)
2. prefetch dell'istruzione target
3. buffer circolare *(loop buffer)*
4. predizione dei salti
5. **salto ritardato** *(delayed branch)*
   - **finche' non si sa** se ci sarà o no il salto (l'istruzione è in pipeline), invece di restare in stallo si può **eseguire un'istruzione che non dipende dal salto**
   - istruzione successiva al salto: ***branch delay slot***
   - Il **compilatore** cerca di allocare nel *branch delay slot* una istruzione "***opportuna***" (magari inutile ma non dannosa)
   - la CPU esegue **sempre** l'istruzione del *branch delay slot* e solo dopo altera, se necessario, la sequenza di esecuzione delle istruzioni

Salto ritardato (delayed branch)

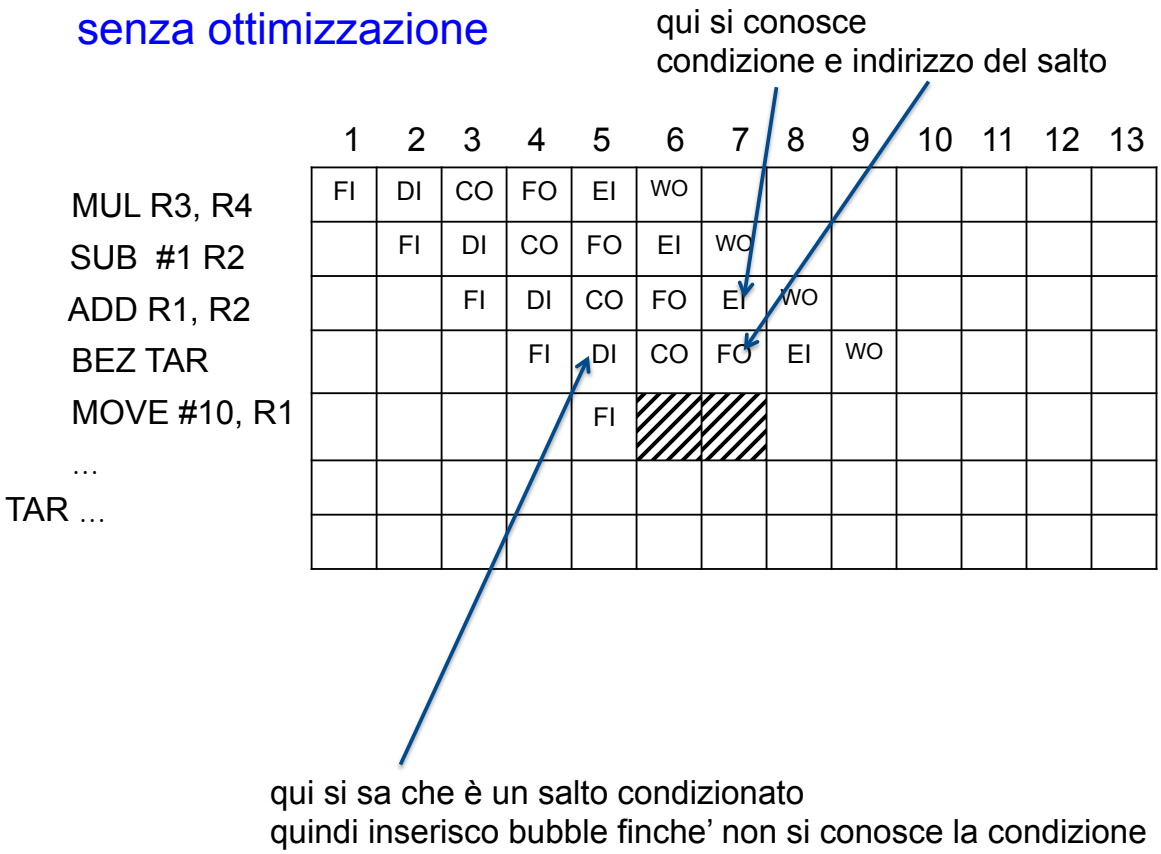
codice scritto dal programmatore



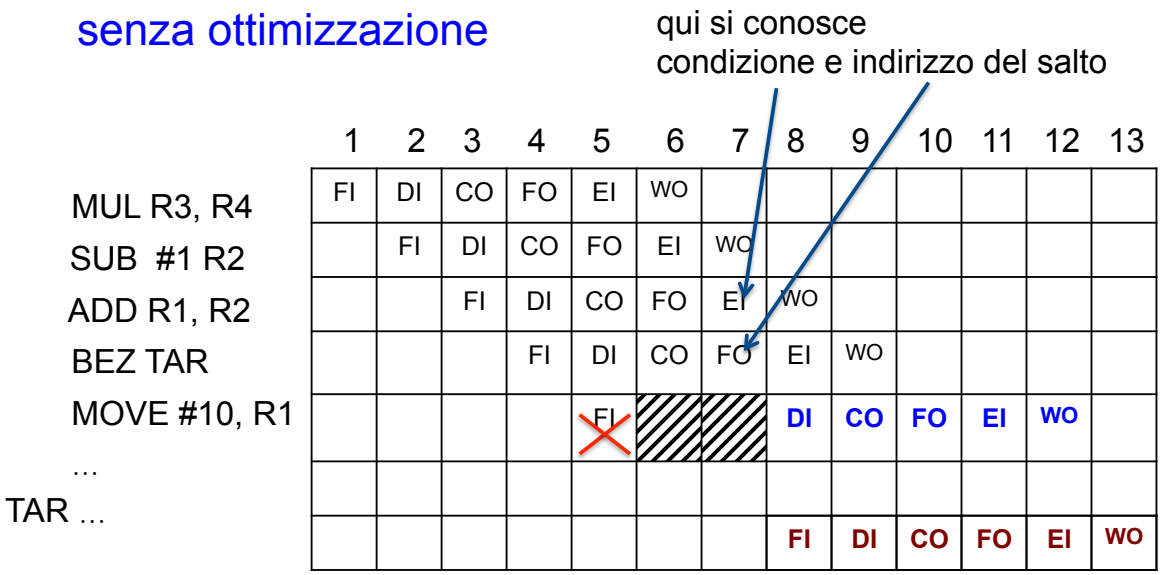
codice ottimizzato dal compilatore



senza ottimizzazione

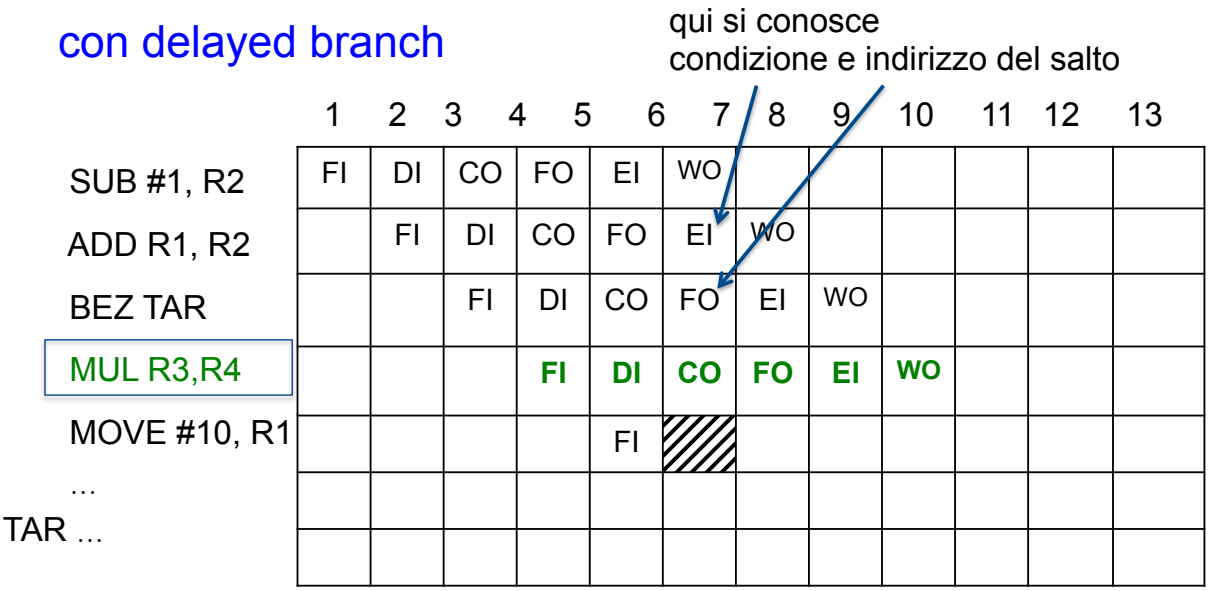


senza ottimizzazione

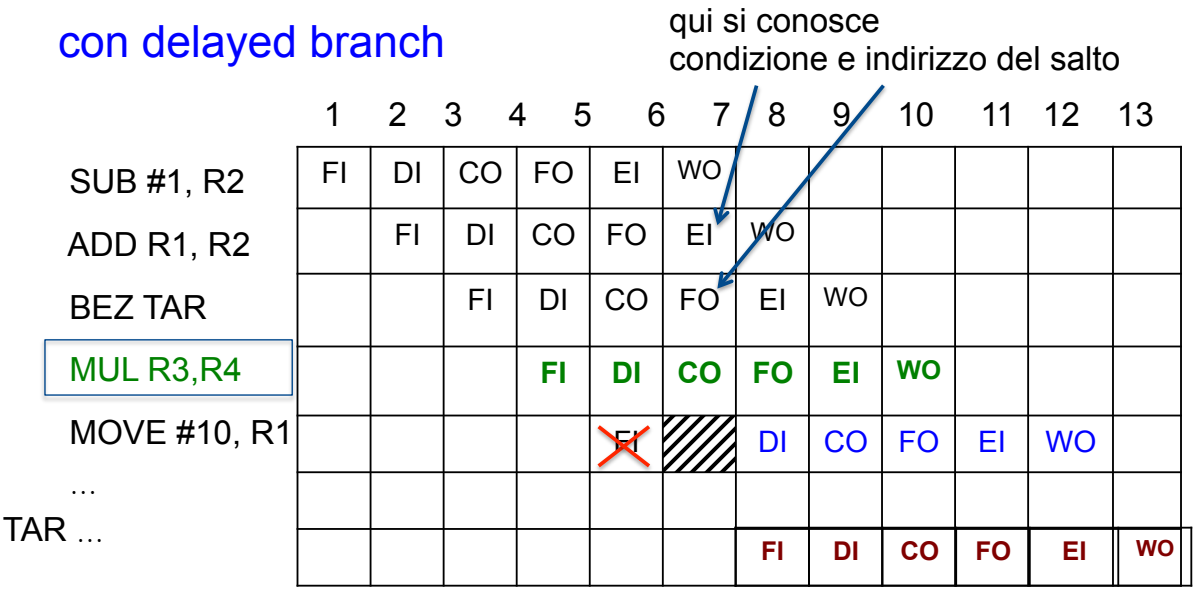


- se **non si salta**
  - continuo con MOVE
  - termino in 12 con **2** cicli di stallo
- se **si salta**
  - scarto MOVE e inizio con TAR
  - termino in 13 con **3** cicli persi (uno inutile 2 stalli)

con delayed branch

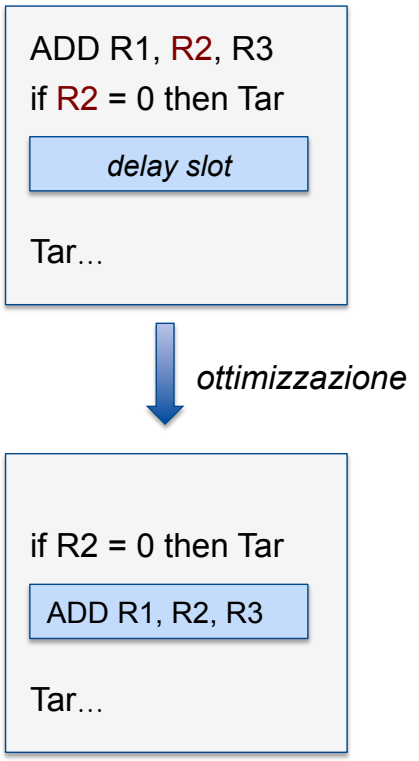


con delayed branch



- se **non si salta**
  - continuo con MOVE
  - termino in 12 con **1** ciclo di stallo
- se **si salta**
  - scarto MOVE e inizio con TAR
  - termino in 13 con **2** cicli persi (uno inutile e 1 stallo)

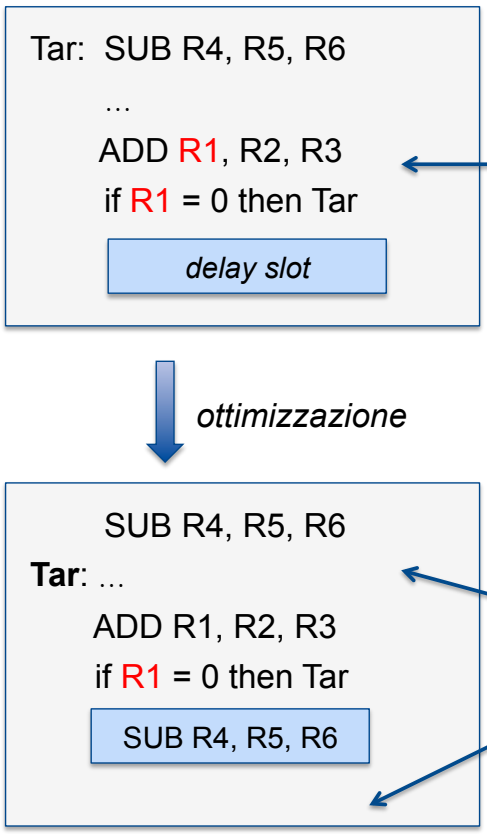
Salto ritardato (delayed branch)



"from before"

quando è possibile riempie il branch delay slot con un'istruzione **indipendente** proveniente dalla parte di codice che *precede* il salto

Salto ritardato (delayed branch)



"from target"

utile quando è probabile che il salto sia preso (es. in loop)

non sono più istruzioni indipendenti

riempie il branch delay slot con **l'istruzione target** del salto, che normalmente viene **copiata** perche' potrebbe essere accessibile anche tramite un altro cammino

eseguo ***sempre*** l'istruzione nel delay slot, quindi:

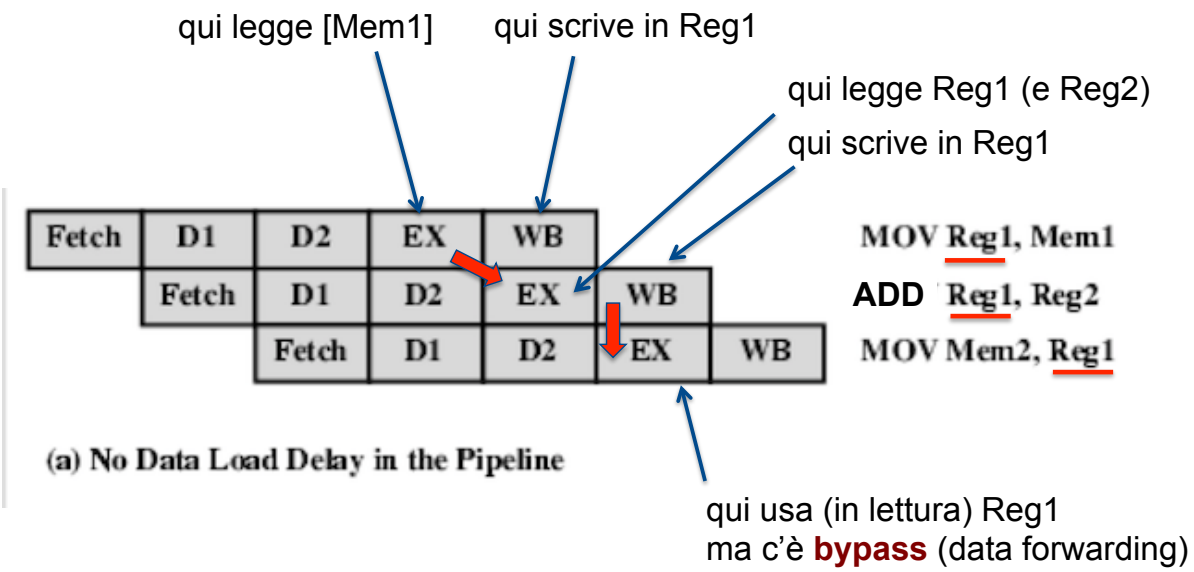
- **quando salto** vado alla successiva
- **quando non salto** procedo oltre, ma è **corretto solo** se **l'istruzione nel delay slot** è inutile ma **non dannosa** (es. R4 non usato se non salta)

Intel 80486 Pipelining

- Fetch
  - Istruzioni prelevate dalla cache o memoria esterna
  - Poste in uno dei due buffer di prefetch da 16 byte
  - Carica dati nuovi appena quelli vecchi sono “consumati”
  - Poiché le istruzioni sono a lunghezza variabile (1-11 byte), in media carica 5 istruzioni per ogni caricamento da 16 byte
  - Indipendente dagli altri stadi per mantenere i buffer pieni
- Decodifica 1 (D1)
  - Decodifica codice operativo e modi di indirizzamento
  - Le informazioni di sopra sono codificate (al più) nei primi 3 byte di ogni istruzione
  - Se necessario, indica allo stadio D2 di trattare i byte restanti (dati immediati e spiazzamento)
- Decodifica 2 (D2)
  - Espande i codici operativi in segnali di controllo per l’ALU
  - Calcola gli indirizzi in memoria per i modi di indirizzamento più complessi
- Esecuzione (EX)
  - Operazioni ALU, accesso alla cache (memoria).
- Retroscrittura (WB)
  - Se richiesto, aggiorna i registri e i flag di stato modificati in EX
  - Se l’istruzione corrente aggiorna la memoria, pone il valore calcolato in cache e nei buffer di scrittura del bus

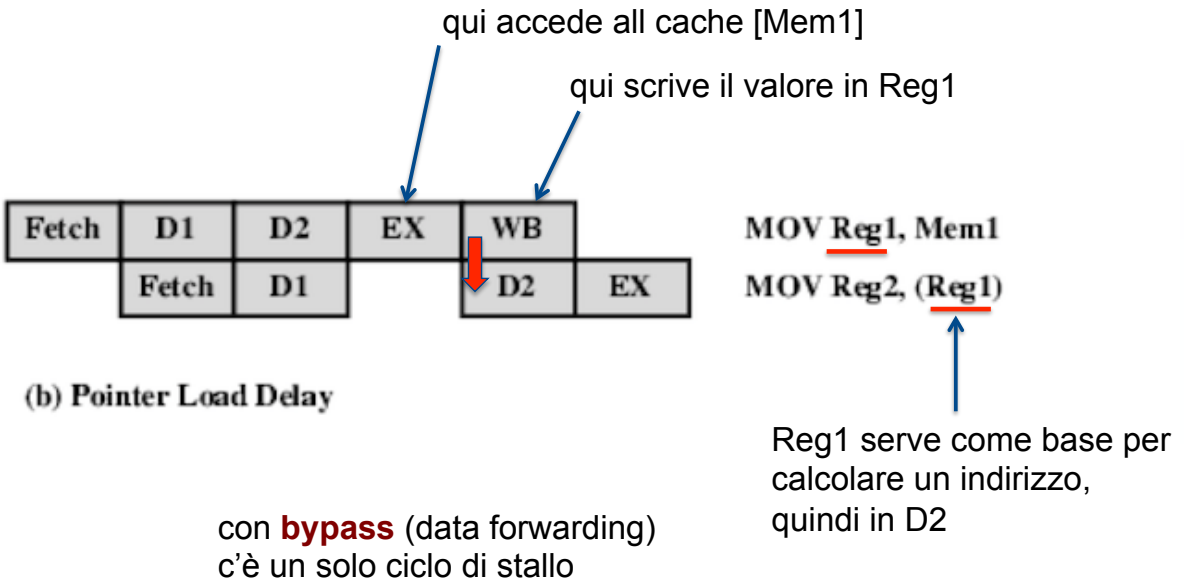
80486 Instruction Pipeline: esempi

accessi consecutivi allo stesso dato non introducono ritardi



80486 Instruction Pipeline: esempi

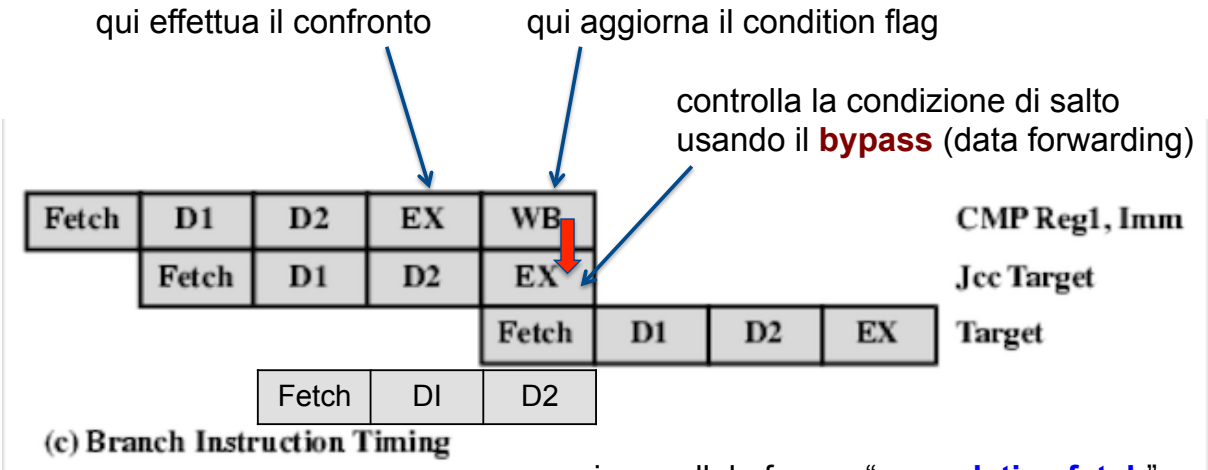
ritardo per valori usati per calcolare un indirizzo



con **bypass** (data forwarding)
c'è un solo ciclo di stallo

80486 Instruction Pipeline: esempi

salto condizionato. Assumiano venga eseguito



in parallelo fa uno "**speculative fetch**" dell'istruzione target
(in aggiunta a quello già iniziato per l'istruzione sequenziale, che sarà scartata).