Pipeline

Francesco Paolo Landolfo 0521000140
Francesca Della Corte 0521000142

Introduzione alle pipeline

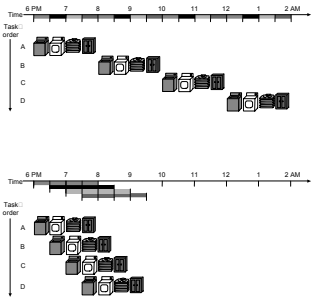
- Le pipeline rappresentano una variante all'unità di elaborazione a singolo ciclo.
- Prevedono la sovrapposizione temporale dell'esecuzione di diverse istruzioni.
- Sono la chiave su cui si basa la velocità dei calcolatori.

Metafora del bucato

- Chiunque abbia avuto a che fare con il bucato ha già utilizzato intuitivamente la tecnica delle pipeline
- Approccio al bucato di tipo classico:
  1. Mettere un carico di abiti sporchi nella lavatrice
  2. Quando la lavatrice ha finito mettere il carico bagnato nell'asciugatrice.
  3. Quando l'asciugatrice ha finito, mettere il carico asciutto sull'asse da stiro e stirarlo.
  4. Dopo aver finito di stirare riporre gli abiti nell'armadio
- Al termine di queste operazioni si potrà ricominciare con il prossimo carico di panni sporchi.

Approccio con pipeline

- Richiede molto meno tempo.
- Appena la lavatrice ha terminato il primo carico e questo è stato trasferito nella asciugatrice, si può riempire la lavatrice con il secondo carico di panni sporchi.
- Quando il primo carico è asciutto, lo si pone sull'asse da stiro per stirarlo, si sposta il carico bagnato nell'asciugatrice ed il successivo carico sporco nella lavatrice.
- Si ripongono nell'armadio gli abiti del primo carico mentre parte la stiratura del secondo carico, con il terzo carico nell'asciugatrice ed il quarto nella lavatrice.
- Risultato: tutti i passi stanno lavorando in contemporanea.



Approccio con pipeline (2)

- Quando si parla di pipeline i passi vengono detti stadi.
- Purchè si abbiano risorse separate per ciascuno stadio, le operazione possono essere poste in pipeline.
- Le pipeline non riducono il tempo necessario per l'esecuzione di un'istruzione, ma semplicemente velocizzano le esecuzioni ponendole in parallelo.
- L'efficacia delle pipeline si apprezza maggiormente quando si ha a che fare con un gran numero di istruzioni.

Istruzioni MIPS

- Le istruzioni MIPS richiedono tipicamente 5 passi:
  1. Prelievo dell'istruzione dalla memoria
  2. Lettura dei registri e decodifica dell'istruzione
  3. Esecuzione dell'operazione o calcolo dell'indirizzo
  4. Accesso ad un operando nella memoria dati
  5. Scrittura del risultato in un registro
- Idea: mettere in parallelo l'esecuzione dei singoli passi relativi a più istruzioni.
- Risultato: la nostra pipeline avrà 5 stadi.

Set di istruzioni orientati alle architetture con pipeline

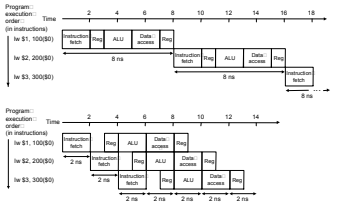
- Il MIPS è stato progettato espressamente per l'esecuzione con processori dotati di pipeline.
- Tutte le istruzioni MIPS hanno la stessa lunghezza: ciò semplifica il prelievo delle istruzioni e la loro decodifica.
- Numero ridotto di formati delle istruzioni e posizione costante dei registri sorgente: permette al secondo stadio di leggere il register file mentre il circuito determina il tipo dell'istruzione.
- Gli operandi di tipo memoria possono comparire soltanto nelle istruzioni di load e di store.
- Si può usare lo stadio di esecuzione per calcolare l'indirizzo di memoria accedendo alla memoria nello stadio successivo.

Criticità sull'uso di pipeline

- Situazioni in cui l'istruzione successiva non può essere eseguita nel ciclo di clock seguente.
- Esistono tre tipi di criticità:
  - Criticità strutturali;
  - Criticità sul controllo;
  - Criticità sui dati.

Criticità strutturali

- Le risorse hardware presenti non sono in grado di supportare la combinazione di istruzioni che si vorrebbe eseguire nello stesso ciclo di clock.
- Analogia con il bucato: un'unica macchina lavatrice/asciugatrice anziché due macchine separate.
- Pipeline: disporre di una sola memoria anziché di due.



Se vi fosse una quarta istruzione, nello stesso ciclo di clock la prima istruzione dovrebbe accedere ai dati nella memoria, mentre la quarta istruzione dovrebbe essere reperita dalla memoria stessa.

Criticità sul controllo

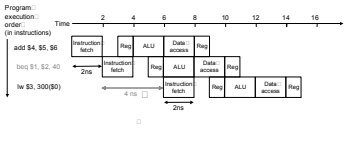
- Necessità di dover compiere alcune scelte in funzione del risultato di un istruzione mentre altre istruzioni sono in esecuzione.
- Analogia con il bucato: lavare le divise di una squadra di calcio e, a seconda di quanto siano sporche, determinare la quantità di detersivo e la temperatura dell'acqua.

Necessario attendere il completamento del secondo stadio

- Esistono due soluzioni alle criticità sul controllo:
  1. Stallo;
  2. Predizione.

Criticità sul controllo - Stallo

- Analogia con il bucato: lavorare in modo sequenziale fino a quando il primo carico è asciutto e se il risultato ottenuto non è quello desiderato ripetere il ciclo finché non si ottiene la combinazione corretta. Corretta ma lenta.
- L'equivalente nel nostro caso è l'istruzione di salto condizionato.
- Sospendere l'esecuzione mediante pipeline procedendo in modo sequenziale fino a terminare l'istruzione di salto.



L'istruzione lw, eseguita se il salto non viene effettuato, viene posta in stato d'attesa per un ciclo di clock della durata di due nanosecondi. Rallentamento ancora maggiore se l'esito dell'istruzione di salto non si può decidere nel secondo stadio.
Costo troppo elevato

Criticità sul controllo - Predizione

- Analogia con il bucato: se si è quasi sicuri di conoscere la giusta combinazione per lavare le divise, si assume di essere in grado di fare una predizione corretta e si lava il secondo carico mentre il primo si asciuga:
  - Predizione corretta: nessun rallentamento;
  - Predizione errata: ripetere la lavorazione del carico che era stato lavato sulla base della predizione.
- Tecnica effettivamente utilizzata dai calcolatori per gestire i salti condizionati. Esistono diversi approcci:
  - **Predire sempre che il salto non venga eseguito:** in caso di predizione corretta la pipeline procede al massimo della velocità, altrimenti si verifica uno stallo;
  - **Predire alcuni salti come eseguiti ed altri come non eseguiti:** ad esempio si può predire che i salti verso indirizzi minori verranno eseguiti (rappresentano dei cicli);
  - **Predizione dinamica:** scegliere la predizione da fare in funzione del comportamento di ciascuna istruzione di salto sulla base di uno storico.

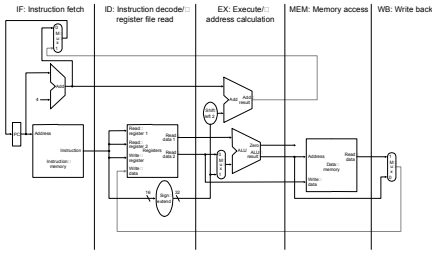
Criticità sui dati

- Analogia con il bucato: si supponga di dover riporre un carico che è composto prevalentemente da calzini e che il calzino corrispondente a ciascuno di quelli del carico corrente è stato posto in un altro carico ancora nella lavatrice

  ⇩

  I calzini non possono essere riposti nell'armadio fino a che il secondo carico non è terminato.
- Nel nostro caso, un'istruzione dipende dal risultato di un'istruzione precedente che si trova ancora nella pipeline
  - Es: add $s0,$t0, $t1
    sub $t2, $s0, $t3
    l'istruzione add non scrive il proprio risultato prima del 5° stadio per cui si dovrebbero aggiungere tre buchi alla pipeline.
- Soluzione più utilizzata → non è necessario attendere il termine dell'istruzione:
  - Non appena la ALU genera il risultato della somma questo viene usato come ingresso per l'operazione di sottrazione. Si prende in anticipo il dato mancante attingendo alle risorse interne.(**Propagazione o scavalcamento**)

Unità di elaborazione con pipeline



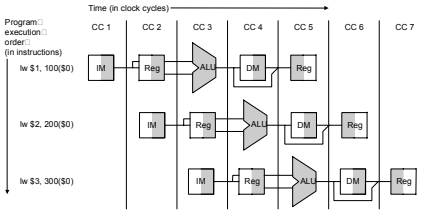
- Nell'immagine è riportata l'unità di elaborazione a singolo ciclo.
- La suddivisione di un'istruzione in 5 fasi implica una pipeline a 5 stadi.

Stadi della pipeline

- Occorre separare l'unità di elaborazione in 5 parti:
  - **IF:** prelievo dell'istruzione;
  - **ID:** decodifica dell'istruzione;
  - **EX:** esecuzione o calcolo dell'indirizzo;
  - **MEM:** accesso alla memoria dei dati;
  - **WB:** scrittura del risultato.
- Durante l'esecuzione istruzioni e dati si spostano, generalmente, da sinistra a destra attraverso le 5 fasi.
- Vi sono due eccezioni al flusso di istruzioni:
  - Lo stadio di scrittura del risultato pone il risultato nel register file;
  - La selezione del valore successivo del PC, scelto tra il valore del PC incrementato e l'indirizzo di salto(proveniente dallo stadio MEM).

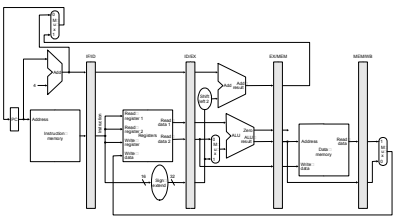
Cosa succede nell'esecuzione con pipeline?

- Ipotizziamo che ciascuna istruzione disponga della propria unità di elaborazione, sistemando tale unità lungo l'asse dei tempi per visualizzarne le relazioni.



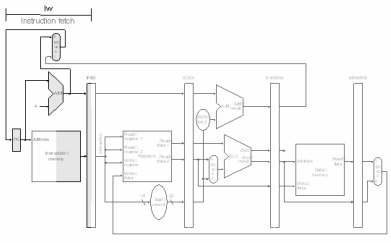
Cosa succede nell'esecuzione con pipeline?(2)

- Per memorizzare il valore dell'istruzione letta dalla memoria a beneficio degli altri 4 stadi, occorrerà salvarlo in un registro.
- Considerazioni analoghe valgono per tutti gli stadi della pipeline.
- Per questo motivo occorrerà aggiungere dei registri ogniqualvolta vi siano delle linee di divisione tra gli stadi.
- Analogia con la lavanderia: cestino tra ciascuna coppia di stadi per contenere i panni destinati al passo successivo.



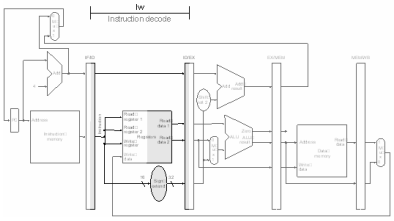
Stadi pipeline

- **Prelievo dell'istruzione:** l'istruzione viene letta dalla memoria utilizzando PC come indirizzo e viene posta nel registro di pipeline IF/ID. L'indirizzo contenuto nel registro PC viene incrementato di 4 e riscritto sia in PC che nel registro di pipeline IF/ID (potrebbe essere richiesto da un istruzione successiva es. BEQ).



Stadi pipeline (2)

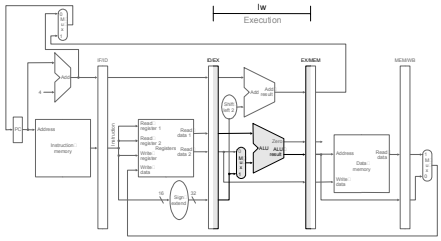
- **Decodifica dell'istruzione e lettura del register file:** la parte inferiore del registro di pipeline IF/ID fornisce il campo di 16 bit ed i numeri dei due registri da leggere. I tre valori sono memorizzati nel registro di pipeline ID/EX insieme al valore incrementato di PC.



- **NOTA: questi due stadi sono comuni a tutte le istruzioni!!**

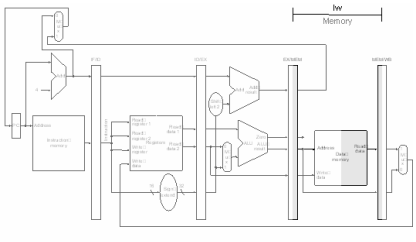
Stadi pipeline – load

- **Esecuzione o calcolo dell'indirizzo:** l'istruzione load legge dal registro di pipeline ID/EX il contenuto del registro 1 ed il valore immediato, esteso con segno, e li somma utilizzando la ALU. Il valore della somma è posto nel registro di pipeline EX/MEM



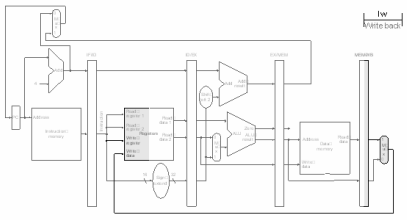
Stadi pipeline – load

- **Accesso alla memoria:** l'istruzione load legge la memoria dei dati utilizzando come indirizzo il valore letto dal registro di pipeline EX/MEM. Il dato letto viene caricato nel registro di pipeline MEM/WB.



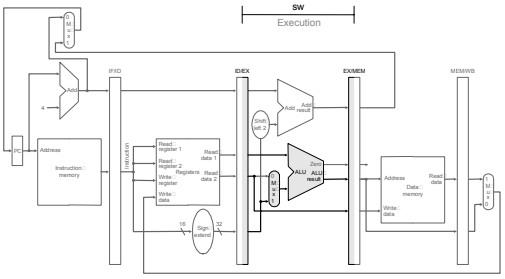
Stadi pipeline – load

- **Scrittura del risultato:** il passo finale è la lettura del dato dal registro di pipeline MEM/WB e la scrittura dello stesso nel register file



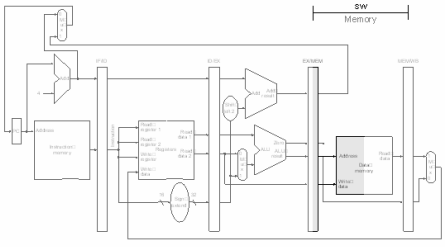
Stadi pipeline - store

- **Esecuzione e calcolo dell'indirizzo:** l'indirizzo dell'operando viene posto nel registro di pipeline EX/MEM



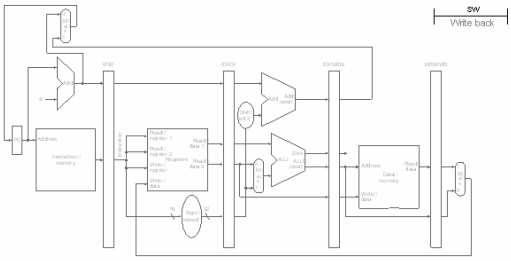
Stadi pipeline - store

- **Accesso alla memoria:** il registro che contiene il dato da dover scrivere è stato letto in uno stadio precedente ed il relativo valore memorizzato nel registro ID/EX. L'unico modo per rendere il dato disponibile durante lo stadio MEM è quello di memorizzarlo nel registro di pipeline EX/MEM durante lo stadio EX.



Stadi pipeline - store

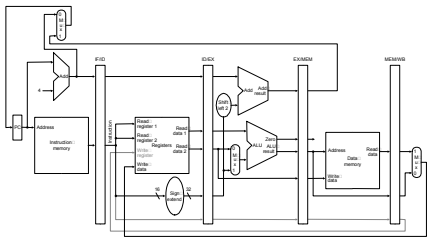
- **Scrittura del risultato:** Nel caso della store non succede nulla!!!! Un' istruzione deve passare in uno stadio anche se non vi è nulla da fare poiché le istruzioni successive stanno già procedendo alla massima velocità possibile.



25

Stadi pipeline - conclusioni

- Per passare un valore da uno stadio iniziale della pipeline ad uno successivo è necessario memorizzarlo in un registro di pipeline, altrimenti l'informazione verrebbe persa.
- Ciascun componente logico dell'unità di elaborazione può essere usato in un solo stadio della pipeline, altrimenti si avrebbe una criticità strutturale.
- Errore nel progetto dell'istruzione load:
  - Chi ci dice il numero del registro da scrivere?
  - Così come la store ha inviato il contenuto del registro dal registro di pipeline ID/EX al registro EX/MEM, anche la load deve fare lo stesso per il numero del registro affinché possa utilizzarlo nello stadio WB.



26

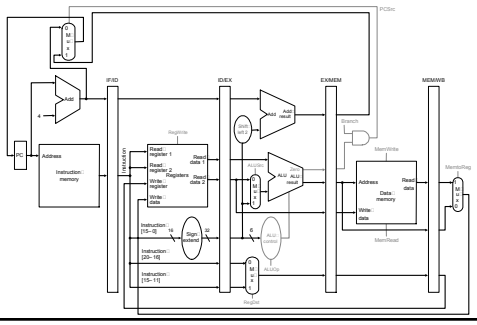
Unità di controllo con pipeline

- Vogliamo aggiungere l'unità di controllo all'unità di elaborazione con pipeline.
- Si partirà con un progetto semplificato che ignora la reale complessità del problema.
- Si approfondirà l'analisi prendendo in considerazione le difficoltà esistenti nel progetto di un sistema reale.

27

Realizzazione dell'unità di controllo

- Il primo passo consiste nell'etichettare i segnali di controllo dell'unità di elaborazione esistente.



28

Realizzazione dell'unità di controllo (2)

- Si assume che il registro PC venga scritto ad ogni ciclo di clock → non vi sarà alcun segnale di controllo per la scrittura di tale registro.
- Non vi saranno segnali di scrittura espliciti per i registri di pipeline, essendo scritti ad ogni ciclo di clock.
- Per definire l'unità di controllo con pipeline è sufficiente calcolare i valori dei segnali di controllo in ciascuno stadio.
- Poiché ciascuna linea di controllo è associata ad un componente e questo è attivo in un solo stadio della pipeline si possono dividere i segnali di controllo in 5 gruppi, secondo il relativo stadio.

29

Realizzazione dell'unità di controllo (3)

- **Prelievo dell'istruzione:** i segnali di controllo per leggere la memoria delle istruzioni e per scrivere il PC sono sempre affermati → nulla da controllare in questo stadio.
- **Decodifica dell'istruzione/lettura del register file:** ad ogni ciclo di clock avviene sempre la stessa cosa → non vi sono segnali di controllo da calcolare.
- **Esecuzione/calcolo dell'indirizzo:** i segnali da calcolare sono RegDst, ALUOp e ALUSrc; essi selezionano il registro risultato, l'operazione della ALU e scelgono il dato da inviare alla ALU tra Dato letto 2 e l'immediato esteso con segno.

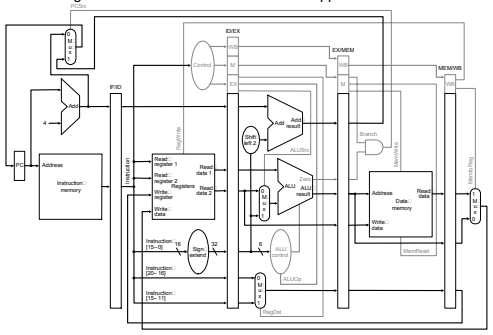
30

Realizzazione dell'unità di controllo (4)

- **Accesso alla Memoria:** i segnali da calcolare sono Branch, MemRead e MemWrite (affermati, rispettivamente, nelle istruzioni branch equal, load e store). Il segnale PCSrc seleziona l'indirizzo successivo a meno che non si affermi il segnale Branch ed il risultato della ALU valga 0.
- **Scrittura del risultato:** i segnali di controllo da calcolare sono MemToReg (sceglie se inviare al register file l'uscita della ALU oppure il valore proveniente dalla memoria) e RegWrite (scrive il valore selezionato).
- Per implementare l'unità di controllo occorre forzare i nove segnali identificati ai valori corretti in ciascuno stadio e per ciascuna istruzione → estendere i registri di pipeline in modo da includere le informazioni di controllo.

Realizzazione dell'unità di controllo (5)

- La seguente figura riporta l'unità di elaborazione completa dei registri di pipeline estesi e dei segnali di controllo connessi allo stadio opportuno:



Criticità sui dati e propagazione

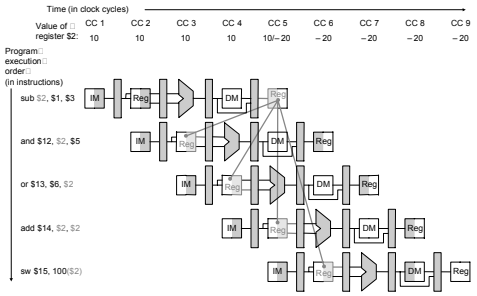
- Nel progetto iniziale di unità di controllo per pipeline non è stato previsto il caso in cui un'istruzione necessiti dei risultati calcolati da un'altra ancora in esecuzione.
- Ecco una sequenza di istruzioni con molte dipendenze:

```
sub $2, $1, $3      # sub scrive nel registro $2
and $12, $2, $5     # il primo operando $2 dipende da sub
or $13, $6, $2      # il secondo operando $2 dipende da sub
add $14, $2, $2     # il primo ed il secondo operando dipendono da
                      sub
sw $15 100 ($2)  # la base più $2 dipende da sub
```

- Come si comporterebbe la sequenza se venisse eseguita sulla pipeline descritta????

Analisi delle dipendenze

- Si avrebbe una situazione simile a quella descritta in figura:



Come risolvere le criticità

- Progettare opportunamente l'hardware che implementa il register file.
- Quando un registro viene letto e scritto nello stesso ciclo di clock si deve garantire che la scrittura avvenga nella prima metà e la lettura nella seconda metà → la lettura fornisce il valore che è stato scritto, risolvendo la criticità sui dati.
- Ritornando al nostro esempio, notiamo che i valori letti dal registro $2 non possono essere quelli prodotti dall'istruzione sub, a meno che l'operazione di lettura non avvenga durante il quinto ciclo di clock o nei successivi → **add** ed **sw** leggono il valore corretto, mentre **and** ed **or** quello scorretto.

Come risolvere le criticità (2)

- È necessario accorgersi della criticità per poi propagare il valore corretto in avanti in modo da risolvere la criticità.
- Quando un'istruzione cerca di leggere nel proprio stadio EX un registro che una precedente istruzione scrive nel proprio stadio WB, è necessario disporre dei valori sugli ingressi della ALU.
- Utilizzare una notazione che assegna dei nomi ai campi dei registri di pipeline.
  - Esempio: ID/EX.RegistroRs fa riferimento al numero di un registro il cui valore si trova nel registro di pipeline ID/EX.
  - La prima parte del nome indica il registro di pipeline.
  - La seconda parte corrisponde al nome del campo all'interno del registro.

Come risolvere le criticità (3)

- Pertanto, utilizzando questa notazione le due coppie di condizioni che generano una criticità si possono esprimere come segue:
  - EX/MEM.RegistroRd = ID/EX.RegistroRs
  - EX/MEM.RegistroRd = ID/EX.RegistroRt
  - MEM/WB.RegistroRd = ID/EX.RegistroRs
  - MEM/WB.RegistroRd = ID/EX.RegistroRt
- Nel nostro esempio, la prima criticità era relativa al registro $2 e si verificava tra le scrittura del risultato dell'istruzione **sub** e l'operazione di lettura del primo operando dell'istruzione **and**.
- Tale criticità può essere rilevata quando l'istruzione **and** si trova nello stadio EX e l'istruzione precedente si trova nello stadio MEM → criticità del primo tipo.

37

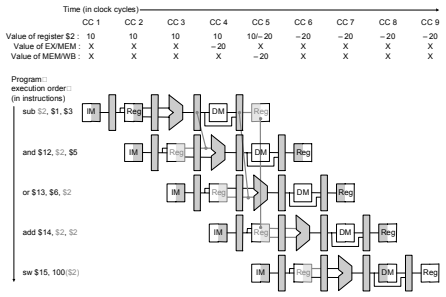
Come risolvere le criticità (4)

- Poiché non tutte le istruzioni scrivono nei registri questa soluzione non è precisa → propaga dei valori anche quando non è necessario.
- Soluzione: verificare se il segnale RegWrite è attivo.
- Ora che siamo in grado di rilevare le criticità metà del problema è risolto: si deve solo cercare di propagare i valori corretti.
- Le istruzioni successive devono prendere il valore di cui necessitano direttamente dai registri di pipeline e darlo in ingresso alla ALU → non è affatto necessario dover attendere lo stadio WB.

38

Propagazione dei valori

- I dati necessari sono disponibili in tempo utile per le istruzioni successive ed i registri di pipeline contengono i dati che devono essere propagati.



39

Propagazione dei valori (2)

- Se gli ingressi della ALU possono essere presi da qualunque registro di pipeline, e non solo da ID/EX, è possibile propagare i dati corretti.
- Aggiungere dei multiplexer sugli ingressi della ALU e utilizzare gli opportuni segnali di controllo. In questo modo sarà possibile scegliere tra i valori provenienti dal register file e quelli propagati.
- Risultato: la pipeline funziona alla velocità massima anche in presenza di dipendenze!!!

40

Criticità EX: condizioni e segnali

- Condizioni:
  - (EX/MEM.RegWrite)
  - EX/MEM.RegistroRd ≠ 0
  - EX/MEM.RegistroRd = ID/EX.RegistroRs
- Azione:
  - Propaga A = 10
- Condizioni:
  - (EX/MEM.RegWrite)
  - EX/MEM.RegistroRd ≠ 0
  - EX/MEM.REgistroRd = ID/EX.RegistroRt
- Azione:
  - Propaga B = 10

Il Multiplexer viene pilotato in maniera da prelevare il valore dal registro di pipeline EX/MEM

41

Criticità MEM: condizioni e segnali

- Condizioni:
  - (MEM/WB.RegWrite)
  - MEM/WB.RegistroRd ≠ 0
  - MEM/WB.REgistroRd = ID/EX.RegistroRs
- Azione:
  - Propaga A = 01
- Condizioni:
  - (MEM/WB.RegWrite)
  - MEM/WB.RegistroRd ≠ 0
  - MEM/WB.REgistroRd = ID/EX.RegistroRt
- Azione:
  - Propaga B = 01

Il Multiplexer viene pilotato in maniera da prelevare il valore dal registro di pipeline MEM/WB

42

Ancora sulle criticità

- Non vi può essere criticità nello stadio WB, in quanto si assume che il register file fornisca il risultato corretto se l'istruzione nello stadio ID legge lo stesso registro scritto dall'istruzione nello stadio WB.
- Possibile fonte di problema: criticità sui dati tra il risultato di un'istruzione nello stadio WB, il risultato di un'istruzione nello stadio MEM e l'operando sorgente di un'istruzione nello stadio ALU.
- Esempio: si somma un vettore di numeri e si scrive il risultato in un singolo registro → sequenza di istruzioni che leggono e scrivono nello stesso registro.
  - **add $1, $1, $2**
  - **add $1, $1, $3**
  - **add $1, $1, $4**
  - ….

Il risultato viene propagato dallo stadio MEM dal momento che il risultato presente nello stadio MEM è il più recente

Criticità sui dati e situazioni di stallo

- Un caso nel quale la propagazione non può risolvere tutti i problemi è quello in cui un'istruzione tenti di leggere un registro subito dopo un'istruzione di load che scrive nello stesso registro.
- Durante il quarto ciclo, il dato deve ancora essere letto dalla memoria quando la ALU si appresta ad eseguire l'operazione prevista dall'istruzione successiva.
- Qualcosa deve mandare in stallo la pipeline.
- È necessario disporre, oltre che di un'unità di propagazione, anche di una unità di rilevamento delle criticità.
- Questa deve operare durante lo stadio ID inserendo lo stallo tra il caricamento e l'utilizzo.

Criticità sui dati e situazioni di stallo(2)

- Poiché si limita a controllare le istruzioni load, il controllo eseguito dall'unità di rilevamento delle criticità coincide con la seguente semplice condizione:
  - **ID/EX.MemRead** → verifica se l'istruzione è una load, unica a leggere dati dalla memoria.
  - **ID/EX.RegistroRt = IF/ID.RegistroRs** e **ID/EX.RegistroRt = IF/ID.RegistroR** → verificano se il campo registro destinazione della load nello stadio EX coincide con uno dei registri sorgente dell'istruzione nello stadio ID.
- Se la condizione è verificata, l'istruzione viene messa in stallo per un periodo di clock.
- Dopo questo stallo di un ciclo, la logica di propagazione può gestire la dipendenza e l'esecuzione può procedere, senza dover attendere un ulteriore ciclo di clock.

Criticità sui dati e situazioni di stallo(3)

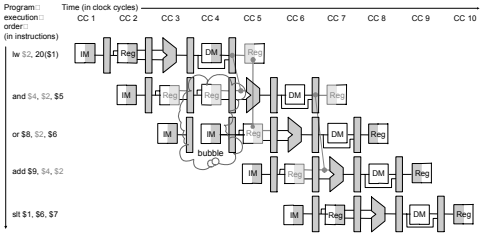
- Se l'istruzione nello stadio ID è messa in stallo, allora anche l'istruzione nello stadio IF deve essere messa in tale stato, altrimenti si perderebbe l'istruzione già caricata.
- Ciò può essere realizzato semplicemente impedendo al registro PC e al registro di pipeline IF/ID di cambiare.
- In questo modo, l'istruzione nello stadio IF continua ad essere letta utilizzando lo stesso PC, ed i registri nello stadio ID continuano ad essere letti utilizzando gli stessi campi istruzione nel registro IF/ID.

Criticità sui dati e situazioni di stallo(4)

- Identificando la criticità nello stadio ID si può inserire nella pipeline un "buco" mettendo a 0 i campi di controllo EX, MEM e WB del registro di pipeline ID/EX.
- Questi valori di controllo si diffondono poi in avanti nella pipeline ad ogni colpo di clock, producendo i loro effetti:
  - Se i valori dei segnali di controllo sono tutti a 0 non si eseguono scritture in alcun registro o parola di memoria.

Criticità sui dati e situazioni di stallo(5)

- Le istruzioni and e or devono ripetere nel ciclo di clock numero 4 ciò che avevano fatto nel ciclo di clock numero 3 → ripetere del lavoro → allungare il tempo delle istruzioni → ritardare il caricamento dell'istruzione add.



In conclusione …

- L'unità di propagazione controlla i multiplexer della ALU per sostituire il valore proveniente da un registro general-purpose con il valore proveniente dall'opportuno registro di pipeline.
- L'unità di rilevamento delle criticità controlla la scrittura del PC e dei registri IF/ID, oltre al multiplexer che sceglie tra i valori reali di controllo e tutti zero.
- L'unità di rilevamento delle criticità mette in stallo e deseleziona i campi di controllo se la condizione di criticità sulla load è verificata.

Criticità sui salti

- Vi sono anche criticità della pipeline che coinvolgono i salti condizionati.
- Per alimentare la pipeline è necessario caricare un'istruzione per ogni colpo di clock, ma nell'attuale progetto le decisioni relative ai salti condizionati vengono presi nello stadio MEM della pipeline.
- Tale ritardo nella determinazione dell'istruzione da caricare viene denominato ***criticità sul controllo*** o ***criticità sui salti condizionati***.
- Tali criticità sono relativamente semplici da capire, si verificano molto meno frequentemente e non vi è nulla che risulti così efficace per la risoluzione delle criticità sul controllo come lo è la propagazione nei confronti delle criticità sui dati.
- Verranno proposti due schemi per la risoluzione delle criticità sul controllo ed un'ottimizzazione che permette di migliorare tali schemi.

Ipotizzare che il salto condizionato non sia eseguito

- Restare in stallo sino a che il salto condizionato non è stato completato fa perdere toppo tempo.
- **Tecnica frequentemente usata**: assumere che il salto non debba essere eseguito e continuare l'esecuzione secondo il normale flusso delle istruzioni.
  - Se poi il salto viene eseguito, allora le istruzioni che si stanno caricando e decodificando devono venire scartate e l'esecuzione riprende dall'istruzione corrispondente alla destinazione del salto.
  - Se i salti non sono eseguiti nella metà dei casi e se costa poco scartare le istruzioni, questa ottimizzazione dimezza il costo delle criticità sul controllo.
- Per scartare le istruzioni ciò che si deve fare è semplicemente cambiare i valori dei segnali di controllo forzandoli a 0.
- Si devono anche cambiare le tre istruzioni presenti negli stadi IF, ID ed EX quando il salto raggiunge lo stato MEM.

Ridurre i ritardi associati ai salti condizionati

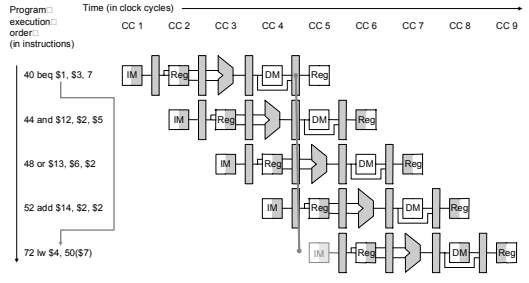
- Un modo per migliorare le prestazioni consiste nel ridurre il costo nel caso in cui il salto venga eseguito.
- Se si sposta l'esecuzione del salto più indietro nella pipeline si riduce il numero di istruzioni che devono essere scartate.
- Finora, si è sempre assunto che il valore da caricare nel PC, nel caso di salto condizionato, venga selezionato nello stadio MEM.
- Si potrebbe risparmiare un ciclo di clock selezionando l'indirizzo di salto già alla fine dello stadio EX.
- Se si potesse spostare la decisione relativa al salto ancora più indietro nella pipeline, allora sarebbe necessario scartare una sola istruzione.
- Molte implementazioni del processore MIPS spostano allo stadio ID l'esecuzione del salto.

Ridurre i ritardi associati ai salti condizionati (2)

- La parte facilmente realizzabile consiste nell'anticipare il calcolo dell'indirizzo di salto.
- Infatti, il valore del PC ed il campo immediato sono già disponibili nel registro IF/ID della pipeline e, quindi, è sufficiente spostare il sommatore per i salti dallo stadio MEM allo stadio ID.
- La parte più difficile consiste nella decisione associata al salto stesso.
- Se il salto è del tipo "salta se uguale" si devono confrontare i due registri letti durante lo stadio ID per verificare se sono uguali.
- Il test di uguaglianza può essere eseguito mettendo innanzitutto in OR esclusivo i bit corrispondenti, e poi mettendo in AND i risultati.
- Tale approccio è molto più veloce da utilizzare rispetto alla ALU (eseguire la sottrazione e poi verificare se l'uscita è nulla).

Effetti delle pipeline sulle istruzioni di salto

- Poiché l'istruzione di salto decide se saltare nello stadio MEM, le tre istruzioni che seguono quella di salto vengono caricate e nell'istante in cui si verifica il salto vengono scartate.

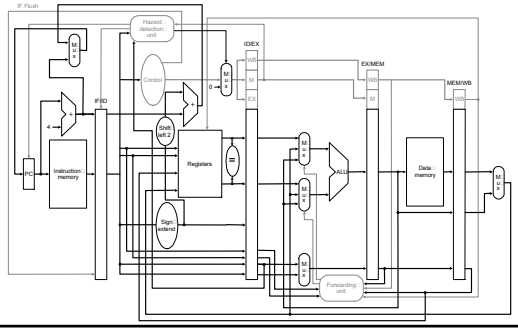


Ridurre i ritardi associati ai salti condizionati (3)

- Spostando l'esecuzione del salto nello stadi ID vi è una sola istruzione da scartare nel casi in cui il salto debba venire eseguito (quella che si trova nella fase di caricamento).
- Per scartare l'istruzione nello stadio IF si aggiunge una linea di controllo, denominata IF.Scarta che azzera il campo istruzione del registro IF/ID della pipeline.
- In questo modo, si trasforma l'operazione appena caricata in una ***nop***, ossia in un'istruzione che non esegue nessuna operazione in grado di cambiare lo stato del sistema.

Unità di elaborazione per i salti

- L'ottimizzazione sposta la decisione relativa al salto dal quarto stadio al secondo



Previsione dinamica dei salti condizionati

- Quando si assume che il salto non debba essere eseguito si realizza una forma rozza di *previsione dei salti*.
- Se si ha a disposizione dell'hardware aggiuntivo è possibile sperimentare altri schemi di previsione dei salti.
- Una possibilità consiste nel verificare l'indirizzo dell'istruzione per controllare se l'ultima volta che si era considerata quell'istruzione il salto fosse stato eseguito:
  - In caso positivo, si caricano le istruzioni successive a partire dalla stessa posizione della volta precedente.
- Implementazione della soluzione: utilizzare un *buffer di previsione dei salti* o *tabella di storia dei salti:*
  - Si tratta di una piccola memoria indicizzata attraverso la parte bassa dell'indirizzo dell'istruzione di salto.
  - Contiene un bit che indica se recentemente il salto è stato eseguito o meno.

Analisi della soluzione proposta

- Si tratta della tipologia di buffer più semplice.
- Non si sa se la previsione è quella giusta:
  - Potrebbe essere relativa ad un altro salto caratterizzato dagli stessi bit per quanto riguarda la parte meno significativa.
- Tutto ciò non ha effetto sulla correttezza dell'operazione:
  - La predizione è soltanto un suggerimento che si assume possa essere corretto, così che il caricamento delle istruzioni possa iniziare con l'indirizzo suggerito.
  - Se il suggerimento si rivela sbagliato, il bit di previsione viene invertito e memorizzato con il nuovo valore, dopodichè si va ad eseguire la sequenza corretta di istruzioni.

Analisi della soluzione proposta (2)

- Lo schema di previsione ad 1 bit ha un difetto in termini di prestazioni:
  - Anche se il salto fosse quasi sempre eseguito, si otterrebbe una previsione scorretta due volte quando il salto non è eseguito.
- Esempio: si consideri un salto condizionato presente all'interno di un ciclo e si supponga che il salto venga eseguito nove volte di seguito e poi non venga eseguito per una volta.
  - Qual è l'accuratezza fornita dalla previsione????
- La previsione non sarà corretta nella prima e nell'ultima iterazione del ciclo.
  - L'errore sull'ultima iterazione è inevitabile in quanto la previsione dirà che il salto deve essere eseguito, essendo stato eseguito nove volte di seguito fino a quel punto.
  - L'errore sulla prima iterazione si verifica perché il bit è stato complementato in occasione dell'esecuzione dell'ultima iterazione del ciclo, quando il salto non era stato seguito.

Analisi della soluzione proposta (3)

- In definitiva, l'accuratezza della previsione per questo salto è pari all'80%.
- Si vorrebbe che per questi salti altamente regolari l'accuratezza della previsione corrispondesse alla frequenza di esecuzione del salto.
- Soluzione: utilizzare schemi di previsione a 2 bit.
  - La previsione deve essere sbagliata per 2 volte prima di essere modificata.
- Un buffer di previsione dei salti può essere realizzato come un piccolo buffer accessibile tramite l'indirizzo dell'istruzione durante lo stadio IF della pipeline.
- Se si prevede che il salto debba essere eseguito, il caricamento viene effettuato a partire dall'indirizzo destinazione non appena è noto il valore del PC.
- Se la previsione si rivela sbagliata, i bit di previsione sono modificati.

Riepilogo sulle pipeline

Obiettivo delle pipeline: *Eseguire un'istruzione per ciclo di clock e permettere un clock più veloce!!!!!*

Eccezioni

- Un'altra forma di criticità sul controllo riguarda le eccezioni.
- Si assuma che la seguente istruzione:
  - Add $1, $1, $1

  generi un overflow aritmetico.
- È necessario trasferire immediatamente il controllo alla procedura di gestione dell'eccezione, dal momento che non si vuole che questo valore non valido contamini altri registri o locazioni di memoria.
- È necessario eliminare dalla pipeline l'istruzione che segue la add ed iniziare il prelievo dell'istruzione successiva a partire dal nuovo indirizzo.

Eccezioni (2)

- Si utilizza lo stesso meccanismo visto precedentemente nel caso dei salti, ma questa volta è l'eccezione a provocare l'azzeramento delle linee di controllo.
- È già stato visto come eliminare un'istruzione nello stadio IF trasformandola in una ***nop.***
- Per eliminare un'istruzione che si trova nello stadio ID si può utilizzare il multiplexer per azzerare i segnali di controllo in caso di stallo.
- Il segnale di controllo ID.Scarta è messo in OR logico con il segnale di stallo che proviene dall'unità di rilevamento delle criticità.

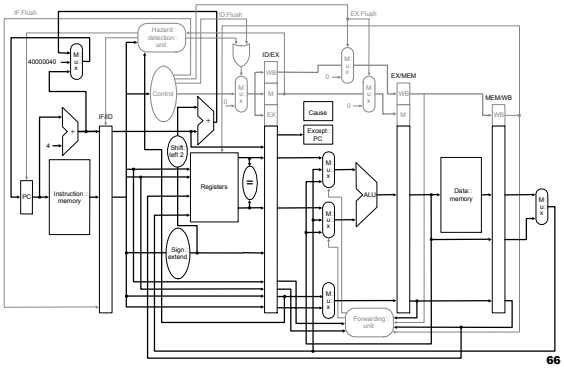
Eccezioni (3)

- Per eliminare l'istruzione nella fase EX si usa il segnale EX.Scarta, che fa in modo che dei nuovi multiplexer azzerino le linee di controllo.
- Per iniziare il prelievo delle informazioni a partire dalla locazione 4000 0040_{esa} (dove si trova la procedura di gestione dell'eccezione), basta semplicemente aggiungere un altro multiplexer del PC.

Eccezioni (4)

- Problema che si verifica nella gestione delle eccezioni:
  - Se non si interrompe l'esecuzione a metà dell'istruzione, il programmatore non è più in grado di vedere il valore originale del registro $1 che ha contribuito a generare la condizione di overflow, dal momento che tale registro è anche l'operando destinazione dell'istruzione.
  - Si deve riconoscere l'eccezione durante lo stadio EX; così il segnale EX.Scarta può prevenire la scrittura del risultato che avviene nello stadio WB.
- Ultimo passo: salvare l'indirizzo dell'istruzione che ha causato l'eccezione nell' Exception Program Counter.
- In realtà viene salvato l'indirizzo +4 e quindi la procedura di gestione delle eccezioni deve per prima cosa sottrarre 4 al valore memorizzato.

Eccezioni (5)



Eccezioni (6)

- Possibili cause di eccezioni:
  - Richiesta da parte dei dispositivi di input/output;
  - Chiamata di un servizio del sistema operativo da parte di un programma utente;
  - Utilizzo di un'istruzione non definita;
  - Malfunzionamenti hardware;
- Poiché si hanno 5 istruzioni attive in ogni ciclo di clock, il problema consiste nell'associare ogni eccezione all'istruzione che l'ha provocata.
- Inoltre vi è la possibilità che nello stesso ciclo di clock più eccezioni si verifichino simultaneamente.

Eccezioni (7)

- Soluzione più frequentemente usata:
  - Associare una priorità alle eccezioni per poter determinare facilmente quale gestire per prima.
  - Strategia che funziona anche per calcolatori dotati di pipeline.
- Nella maggioranza delle implementazioni MIPS, i circuiti ordinano le eccezioni in modo tale che venga interrotta l'istruzione eseguita per prima fra quelle coinvolte.
- Le richiesta da parte dei dispositivi di ingresso/uscita ed i malfunzionamenti hardware non sono associati ad un'istruzione specifica → consentita una maggiore flessibilità nell'implementazione.
  - i circuiti possono scegliere l'istruzione più semplice da associare all'eccezione;
  - Nel caso di malfunzionamenti hardware è opportuno interrompere l'esecuzione al più presto (hardware instabile!).

Eccezioni (8)

- Il registro EPC mantiene l'indirizzo dell'istruzione interrotta.
- Il registro MIPS Causa memorizza tutte le possibili eccezioni che si verificano in un ciclo di clock.
- Il programma di gestione deve trovare la corrispondenza fra istruzione ed eccezione.
- Per capire l'associazione eccezione-istruzione → stadio della pipeline in cui può verificarsi un certo tipo di eccezione:
  - Un'istruzione non definita è scoperta nello stadio ID;
  - Una chiamata al sistema operativo avviene nello stadio EX.
- Le eccezioni sono collezionata nel registro Causa → i circuiti possono interrompere la normale esecuzione basandosi su eccezioni che si verificano dopo che eccezioni precedenti sono state risolte.

Eccezioni (9)

- Nei calcolatori dotati di pipeline, è molto difficile associare correttamente un'eccezione all'istruzione che l'ha provocata.
- Per questo motivo, molti progettisti hanno rilassato i requisiti nei casi non critici:
  - Si dice che questi calcolatori hanno ***interruzioni imprecise*** o ***eccezioni imprecise***.
- Il MIPS e la stragrande maggioranza dei calcolatori di oggi supportano ***interruzioni precise*** o ***eccezioni precise***.

Evoluzioni delle pipeline

- Nell'intento di realizzare processori sempre più veloci, le semplici pipeline descritte si sono evolute secondo tre direzioni principali:
  - ***Superpipeline***
  - ***Superscalare***
  - ***Schedulazione dinamica delle pipeline***
- ***Superpipeline***: sono delle pipeline più lunghe.
  - Poiché il massimo incremento di velocità di una pipeline ideale è legato al numero degli stadi, alcuni microprocessori recenti hanno pipeline con otto o più stadi.
  - Svantaggio: necessario bilanciare nuovamente i restanti passi, in modo che abbiano la stessa durata.

Evoluzioni delle pipeline (2)

- ***Superscalare***: consiste nel replicare i componenti interni del calcolatore in modo che sia possibile lanciare l'esecuzione di più istruzioni in ogni stadio della pipeline.
  - Svantaggio: lavoro addizionale necessario per mantenere impegnate tutte le macchine e per trasferire il carico di lavoro da uno stadio della pipeline al successivo.
  - Gli attuali calcolatori superscalari cercano di far eseguire da due a sei istruzioni in ogni stadio della pipeline; se però le istruzioni all'interno del flusso hanno delle dipendenze o non soddisfano determinati criteri, soltanto le prime istruzioni vanno a buon fine.

Evoluzioni delle pipeline (3)

- ***Schedulazione dinamica della pipeline***: consiste nell'utilizzo di circuiti logici che evitano il verificarsi di condizioni critiche.
- La pipeline è in condizione di stallo quando aspetta che una criticità sia risolta, anche se le istruzioni successive sono già pronte.
- Le pipeline dinamiche sono normalmente affiancate da circuiti addizionali che fanno si che istruzioni successive possano procedere in parallelo.
- Svantaggi:
  - Unità di controllo delle pipeline molto più complessa.
  - Modello dell'esecuzione delle istruzioni molto più complicato.

MIPS superscalare

- Com'è fatto un calcolatore MIPS con implementazione superscalare?
- Supponiamo di voler eseguire due istruzioni per ciclo di clock.
- È necessario il prelevamento e la decodifica di 64 bit di istruzione.
- Per semplificare la decodifica:
  - Richiedere che le istruzioni siano accoppiate ed allineate a multipli di 64 bit.
  - Richiedere che in ogni coppia di istruzioni la prima sia quella che esegue un'operazione con la ALU o un salto condizionato.
- In alternativa si potrebbero esaminare le istruzioni ed eventualmente scambiarle prima che vengano inviate alla ALU o alla memoria.
- In entrambi i casi la seconda istruzione deve essere presa in considerazione solo se la prima può essere eseguita.

MIPS superscalare (2)

- Per eseguire in parallelo un'operazione logico-aritmetica ed una di trasferimento dati il primo requisito hardware aggiuntivo riguarda la predisposizione di porte addizionali per il register file.
- In un solo ciclo di clock l'operazione della ALU potrebbe dover leggere due registri ed altrettanto potrebbe dover fare l'operazione di trasferimento dati.
- Possono servire due porte in lettura:
  - Una per l'operazione della ALU.
  - Una per l'operazione di trasferimento dati.
- Poiché la ALU è impegnata nell'esecuzione dell'operazione logica o aritmetica è necessario un sommatore separato per il calcolo dell'indirizzo per il trasferimento dati.
- Senza queste risorse addizionali il funzionamento della pipeline superscalare presenterebbe delle criticità di tipo strutturale.

MIPS superscalare (3)

- Ulteriore difficoltà:
  - Nella pipeline MIPS vista le istruzioni load hanno una latenza pari ad un ciclo di clock.
  - Un'istruzione non può usare il risultato senza un'attesa.
  - Nella pipeline superscalare il risultato di un'istruzione load non può essere utilizzato nel ciclo di clock successivo.
  - Se le due istruzioni seguenti devono usare il risultato della load sono costrette ad un periodo di attesa.
- In un processore superscalare, per sfruttare il parallelismo in modo efficace sono necessari:
  - Compilatori più sofisticati.
  - Tecniche di schedulazione hardware.
  - Tecniche più complesse per la decodifica delle istruzioni.
- Una tecnica per ottenere prestazioni migliori da cicli che accedono a vettori è quella dell'***espansione dei cicli***.
- Vengono inserite più copie del corpo del ciclo ed istruzioni appartenenti ad iterazioni diverse sono schedulate insieme.

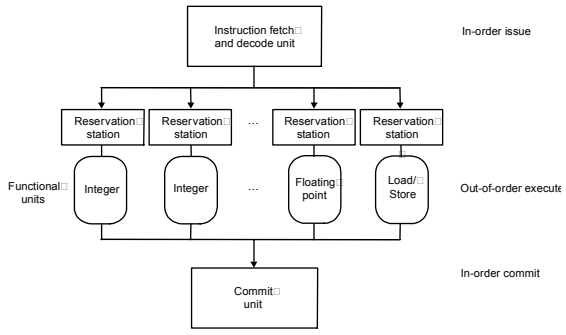
Quadro d'insieme

- Sia le pipeline che l'esecuzione superscalare aumentano il throughput di picco delle istruzioni.
- Svantaggi:
  - Pipeline più lunghe e versioni scalari di maggiore ampiezza rendono il compito del compilatore ancora più complesso, nel tentativo di sfruttare al massimo le potenzialità fornite dall'hardware.
  - Le dipendenze tra i dati e i segnali di controllo all'interno dei programmi, insieme alla latenza delle istruzioni, rappresentano un limite superiore per le prestazioni effettive.
  - Il processore è costretto ad aspettare per via di una dipendenza non ancora risolta o di un salto condizionato non previsto correttamente.
- È comunque necessario garantire la corretta esecuzione di tutte le sequenze di istruzioni.

Schedulazione dinamica della pipeline

- La schedulazione dinamica della pipeline supera lo stallo cercando di eseguire istruzioni successive in attesa che lo stallo sia risolto.
- Tipicamente la pipeline è suddivisa in tre unità principali:
  - Un'unità per il prelevamento e l'invio dell'istruzione.
  - Un'unità di esecuzione.
  - Un'unità di consegna.
- **Prima unità**: preleva le istruzioni, le decodifica ed invia ciascuna di esse alla corrispondente unità funzionale dello stadio relativo all'esecuzione.
- **Seconda unità**: ciascuna unità funzionale ha dei buffer (stazioni di prenotazione) che mantengono operandi ed operazione. Non appena la stazione di prenotazione contiene tutti gli operandi richiesti e l'unità funzionale è pronta per l'esecuzione, viene calcolato il risultato.
- **Terza unità**: decide se salvare il risultato nel register file oppure in memoria.

Unità fondamentali



Schedulazione dinamica della pipeline (2)

- Per fare in modo che i programmi si comportino come se fossero in esecuzione su un semplice calcolatore senza pipeline, l'unità di prelevamento e decodifica deve trattare le istruzioni in ordine sequenziale, e quella di consegna deve scrivere i risultati nei registri o in memoria seguendo l'ordine di esecuzione del programma (***completamento in ordine***).
- Le unità funzionali, invece, sono libere di iniziare e finire l'esecuzione a piacere.

Schedulazione dinamica della pipeline (3)

- Un approccio più radicale consiste nel consentire al calcolatore di fornire i risultati anche in ordine diverso rispetto a quello di esecuzione del programma (***completamento fuori ordine***).
- Le pipeline a schedulazione dinamica sono più complesse:
  - la schedulazione dinamica è spesso abbinata alla predizione dei salti →l'unità di consegna deve essere in grado di scartare tutti i risultati nelle unità di esecuzione dovuti ad istruzioni eseguite in seguito ad un salto erroneamente previsto. Tale combinazione prende il nome di ***esecuzione speculativa***.
  - La schedulazione dinamica è spesso combinata con l'esecuzione superscalare → ciascuna unità può inviare o consegnare da quattro a sei istruzioni per ciclo di clock.

Schedulazione dinamica della pipeline (4)

- I calcolatori dinamici predicono il flusso delle istruzioni, eseguendo le istruzioni in base a congetture fondate sulla predizione e sulle dipendenze esistenti tra le istruzioni stesse.
- Motivazioni alla base dell'esecuzione dinamica:
  - Nascondere la latenza della memoria.
  - Evitare le situazioni di stallo che il compilatore non è in grado di risolvere, dovute a dipendenze potenziali fra istruzioni load e store.
  - Eseguire delle istruzioni, in base a congetture sul flusso futuro, in attesa che le situazioni critiche vengano risolte.
- Il DEC Alpha 21264 è un esempio di microprocessore che realizza tutte e tre le possibilità descritte (pipeline più lunghe, superscalare e schedulata dinamicamente).

Un caso reale: PowerPC 604 e Pentium Pro

- Entrambi utilizzano pipeline schedulate dinamicamente.
- La cache delle istruzioni preleva 16 byte di istruzioni e le invia ad una coda di istruzioni (4 istruzioni per il PowerPC ed un numero variabile per Pentium Pro).
- Un certo numero di istruzioni sono prelevate e decodificate.
- Per prevedere i salti ed eseguire le istruzioni successive entrambi i processori usano una tabella da 512 elementi contenente le informazioni sui salti eseguiti in tempi precedenti.
- L'unità di smistamento invia ciascuna istruzione, con i relativi operandi, alla stazione di prenotazione corrispondente all'unità funzionale prescelta e inserisce un riferimento a quella istruzione all'interno del buffer di riordino dell'unità di consegna → un'istruzione non può essere considerata se non c'è spazio disponibile nella stazione di prenotazione oppure nel buffer di riordino.

Un caso reale: PowerPC 604 e Pentium Pro (2)

- Molte istruzioni vengono eseguite contemporaneamente → è possibile esaurire lo spazio per mantenere i risultati:
  - Entrambi i processori hanno registri interni addizionali (***registri replicati***) usati per memorizzare i risultati in attesa che l'unità di consegna dia il permesso per scrivere i registri effettivi.
  - Questi registri sono assegnati dall'unità di decodifica per ridurre le criticità sul numero di registri.

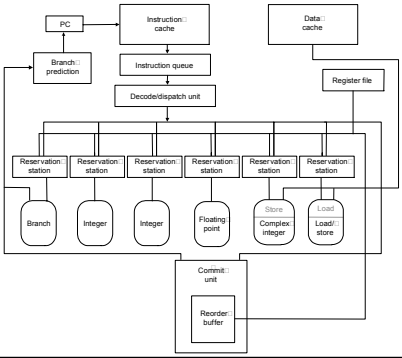
  L'istruzione viene eseguita quando:
  - Il corrispondente elemento all'interno della stazione di prenotazione possiede tutti gli operandi.
  - Le relativa unità funzionale è libera.
- L'unità di consegna tiene traccia di tutte le istruzioni pendenti all'interno del buffer di riordino.
- Poiché utilizzano la predizione dei salti, un'istruzione non è terminata fino a quando non lo decide l'unità di consegna.

Un caso reale: PowerPC 604 e Pentium Pro (3)

- Quando l'unità funzionale che si occupa dei salti decide se un salto deve o non deve essere effettuato, lo comunica all'unità di predizione dei salti (aggiorna lo stato del processore) e all'unità di consegna (decide la sorte delle istruzioni pendenti).
  - ***Predizione corretta***: i risultati delle istruzioni sono confermati e possono essere inseriti nei registri.
  - ***Predizione errata***: i risultati sono invalidati e scartati dalle stazioni di prenotazione e dal buffer di riordino.
- L'unità di consegna può fornire in uscita più istruzioni in un ciclo di clock:
  - Fa in modo che le istruzioni siano consegnate nello stesso ordine in cui sono state inserite nella pipeline.
  - Non può consegnare un'istruzione fino a che l'unità funzionale non ha finito l'operazione corrispondente, tutti i salti condizionati da cui dipende non sono stati risolti e tutte le istruzioni precedenti non state consegnate.

85

Un caso reale: PowerPC 604 e Pentium Pro (3)



86

FINE

87