

Capitolo 4 : Struttura del processore

Prestazioni della CPU

I fattori che influenzano le prestazioni della CPU sono vari; in particolare se volessimo sapere quanto tempo impiega un programma ad essere eseguito, bisogna sicuramente sapere quante istruzioni dovranno essere eseguite, e di conseguenza questo dipende dalla **struttura di processore e compilatore**.

Processori odierni riescono a eseguire, "in un colpo solo", istruzioni molto complesse, anche se queste durano più tempo. Quindi un processore che riesce ad eseguire istruzioni complesse in un'unica volta non detto che sia più veloce di un altro che le esegue in più "parti".

Due versioni del RISC

Vedremo due versioni del RISC V, la prima molto basilare, mentre la seconda più realistica e basata sul tipo di architettura moderna; questa versione è basata sul pipelining.

Quali istruzioni utilizzeremo?

Verranno esaminate solo **alcune** istruzioni del RISC:

- Istruzioni di interfacciamento alla memoria: **ld, sd**
- Istruzioni aritmetiche e logiche: **add, sub, and, or**
- Trasferimento di controllo: **beq**

Esecuzione di un'istruzione

L'esecuzione di un'istruzione è composta da una serie di fasi :

1. Fase di fetch - prelievo dell'istr dalla memoria
2. Fase di preparazione degli operandi
3. Fase di esecuzione

Chi comanda?

Il PC - Program Counter (registro) è il registro che contiene l'indirizzo della prossima istruzione; sulla base di quello che è contenuto da questo registro, si va a prelevare dalla memoria.

Usare l'ALU per calcolare

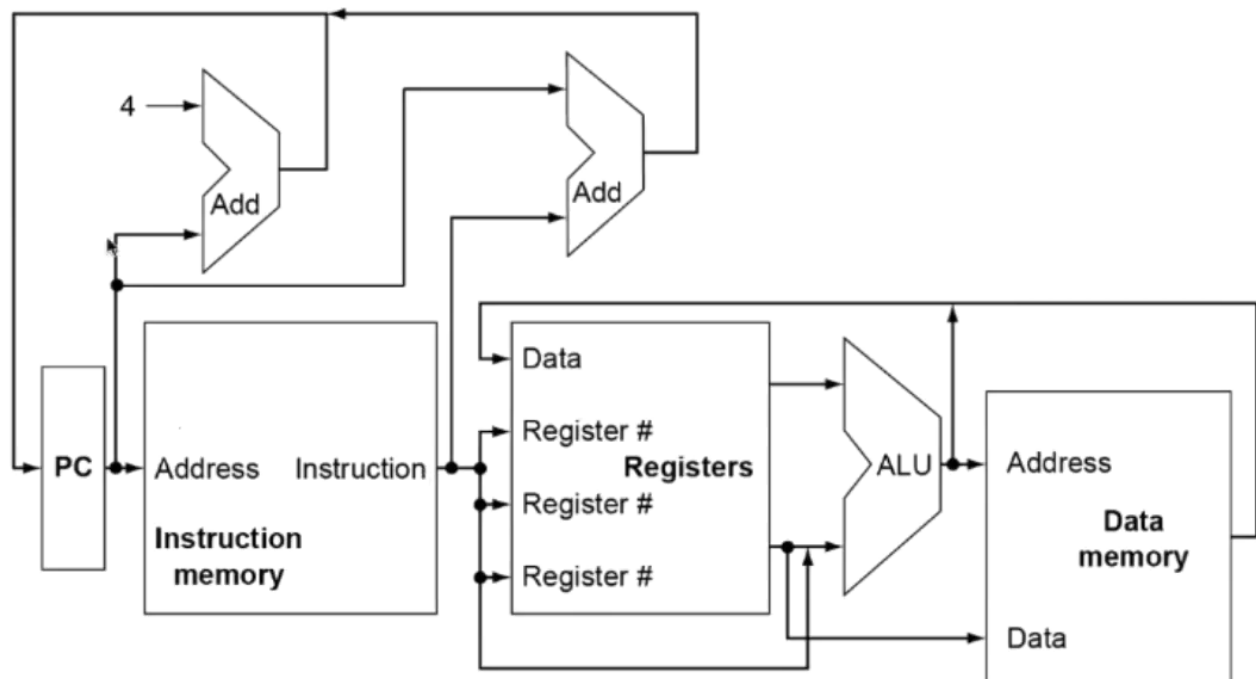
Dopo aver effettuato le fasi preliminari, bisogna usare L'ALU per:

- calcolare un risultato aritmetico
- **Calcolare** l'indirizzo per il load/store; queste istr prevedono un registro più una costante da sommar. Quando trovo un'istruzione di Load/store, sono costretto a sommare al contenuto di un registro, una costante espressa **all'interno dell'istruzione** .
L'ALU serve quindi unicamente a calcolare l'indirizzo.
- Confrontare per un branch.

Overview Della CPU

Abbiamo una memoria da cui vengono prelevate le istruzioni ed una memoria in cui sono presenti i dati. Il PC ci dice l'indirizzo da cui prelevare l'istruzione; L'istruzione viene decodificata per permettere l'accesso ai registri

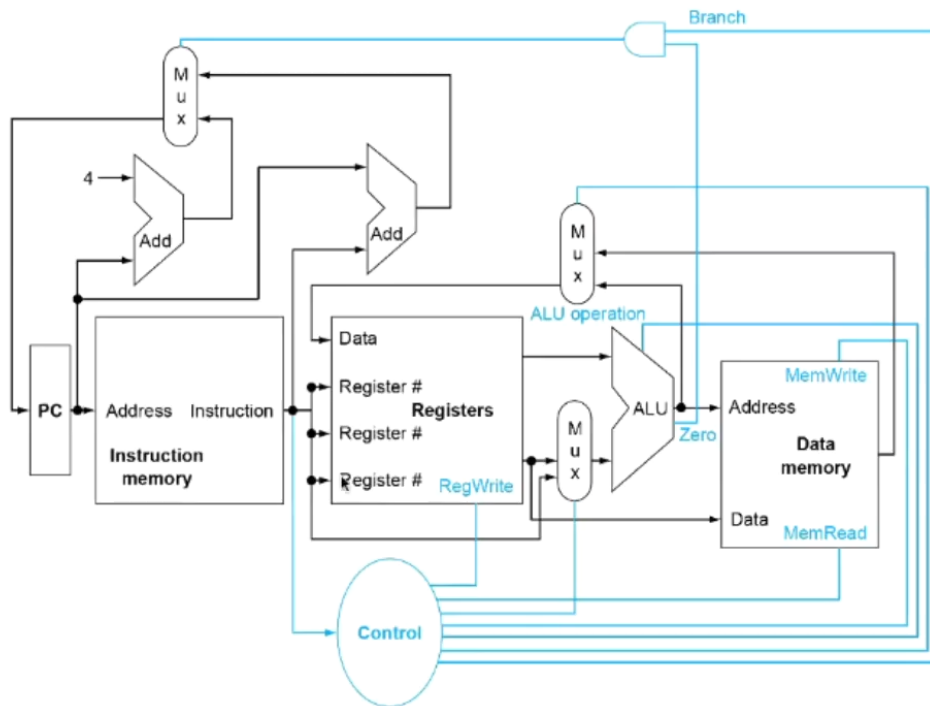
CPU Overview



Negli "incroci" che si notano nell'immagine sono posti dei **multiplexer**, che sotto l'effetto di un **segnale di controllo** decide quale segnale far passare.

Controllo

Control



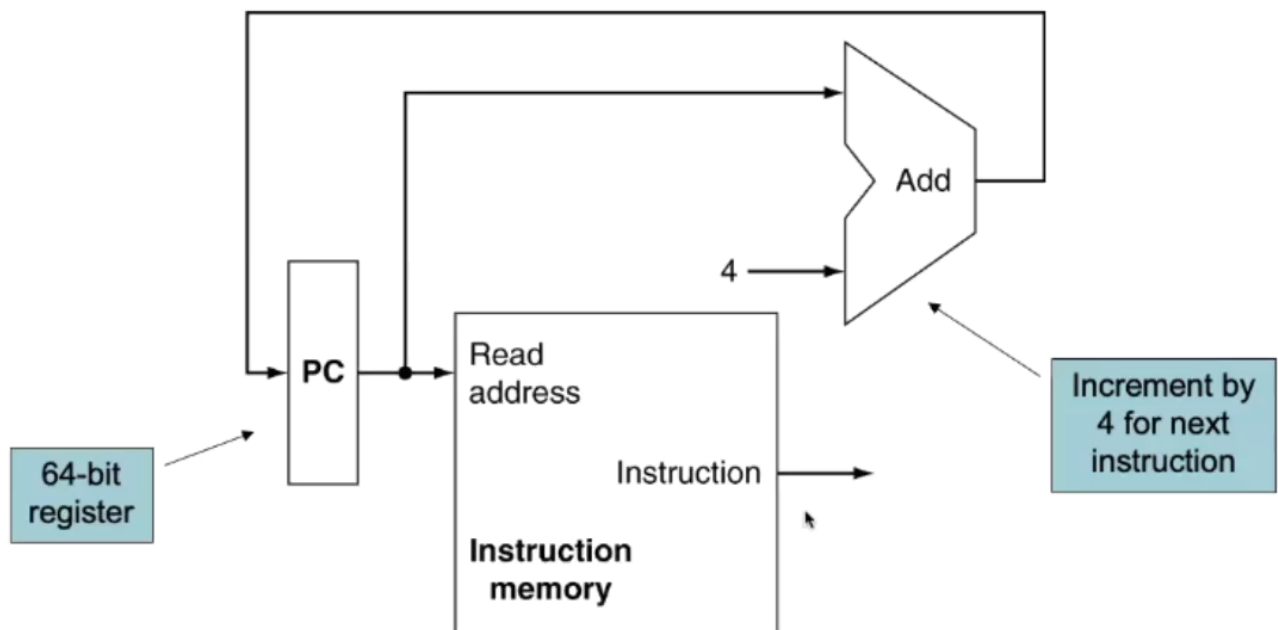
Come vediamo abbiamo un segnale di controllo che abilita o disabilita i multiplexer (ed altro).

Elementi sequenziali

I registri del processore, sono elementi **sequenziali**, memorizzando un risultato precedente, hanno un'uscita che dipende da quello presente precedentemente.

Fetch delle istruzioni

Instruction Fetch



Questa fase avviene prelevando un valore dal PC (registro) che contiene l'indirizzo della prossima istruzione da eseguire. Il PC va fornito al blocco **instruction memory**, fornendo quindi l'indirizzo di lettura.

Il risultato sarà che la memoria delle istruzioni restituirà i bit che compongono quell'istruzione; le istruzioni sono a **32 bit**.

Mentre che la memoria delle istruzioni restituisca il valore richiesto, il segnale del PC va anche in input ad un **adder**, che sommato a 4 (fisso), per incrementare il valore del PC alla prossima istruzione.

Il PC **non verrà aggiornato fino al prossimo ciclo di clock**.

Istruzioni R-Format

Le istruzioni R-Format sono delle istruzioni aventi **3 operandi** di tipo **registro**.

Queste operazioni prendono il contenuto di due registri, eseguono l'operazione (somma, sottrazione, ecc) e salvano il valore in un terzo registro. Per indicare un registro **servono 5 bit**

L'ALU invece, riceve **4 bit che indicano l'operazione da eseguire** e restituisce il risultato; l'ALU potrebbe inoltre restituire **un singolo bit**, che ci dice se l'operazione è zero: questo output è usato nell'istruzione **beq** per vedere se i due operandi sono uguali oppure no.

🚩 1:00 04-08

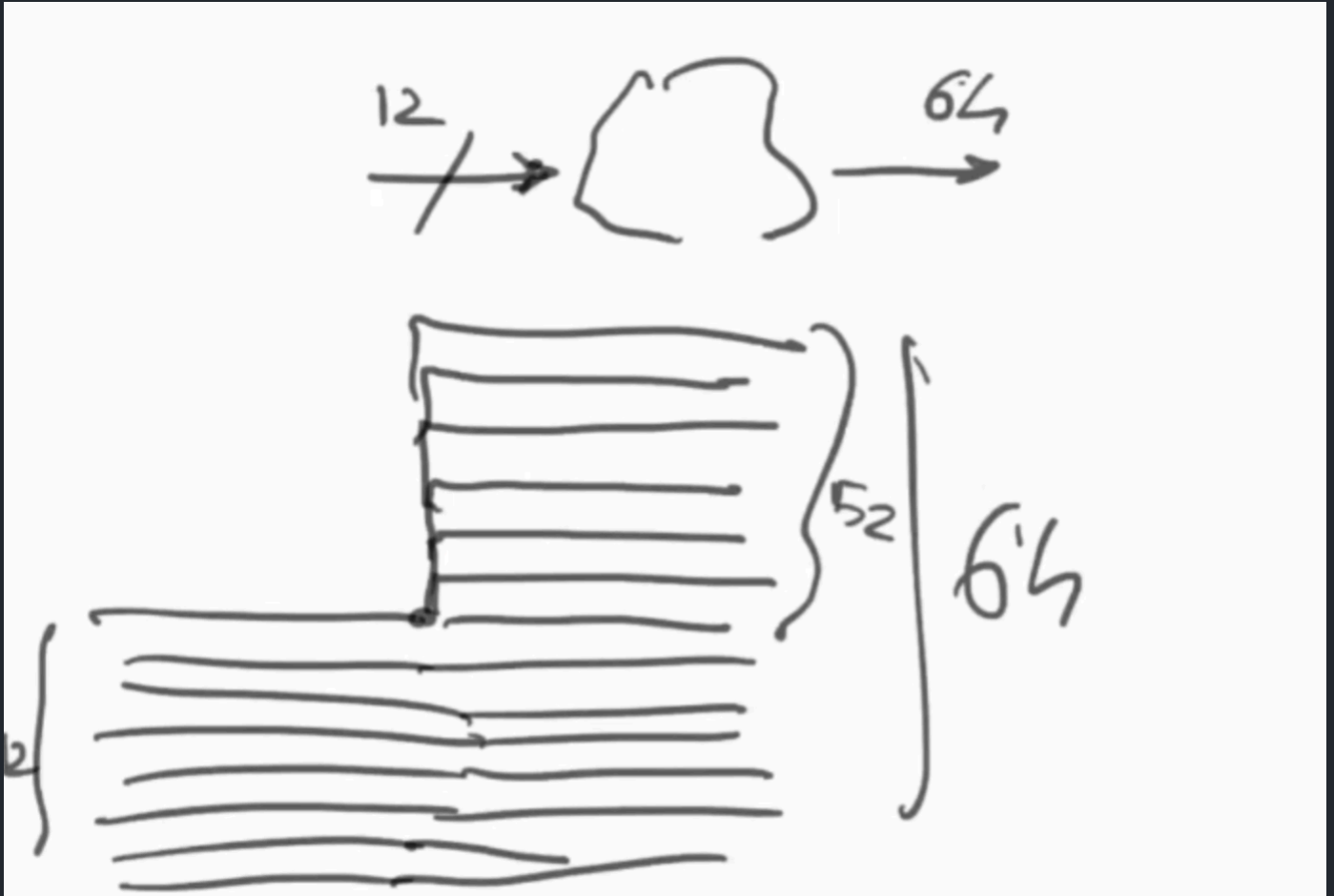
Istruzioni Load/Store

Le istruzioni load store lavorano su un indirizzo fornito da un registro. Quindi nel momento in cui c'è un'istruzione di load store si deve leggere in un **operando di tipo registro** al quale deve essere sommato un offset a 12 bit espresso nell'istruzione.

Ovviamente dobbiamo quindi "far diventare" quei 12 bit 64, quindi abbiamo bisogno di **un'estensione del segno**.

Estensione del segno

E' molto semplice effettuare l'estensione del segno (in binario); ricordiamo che l'operazione consiste nell'estendere **il bit più significativo**.



Opera d'arte omaggiata dal prof

Istruzioni di branch

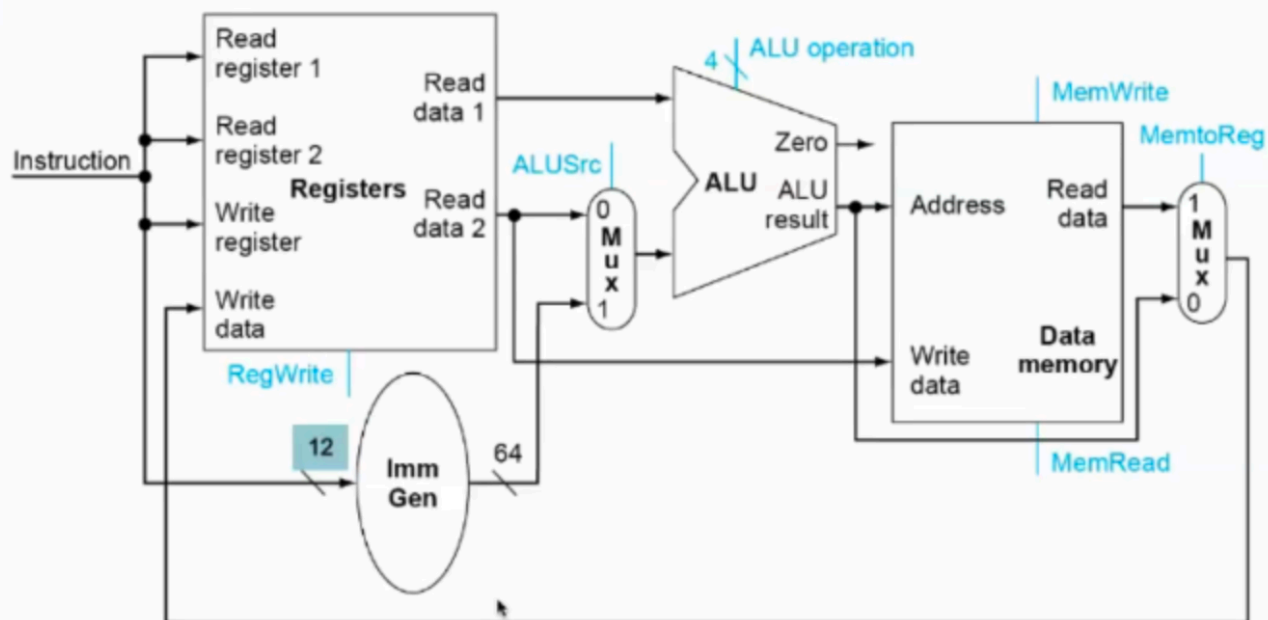
Per vedere se due operandi sono uguali, basta semplicemente usare l'ALU per effettuare una sottrazione. Se la sottrazione risulta zero, gli operandi sono uguali; questo è il caso in cui viene usato il secondo output zero dell'ALU.

Unire i componenti

Il primo data path è in grado di eseguire un'istruzione in un periodo di clock; in pratica durante un periodo di clock i segnali attraversano questa rete e ogni elemento del datapath "lavora" una sola volta.

I multiplexer servono quindi a decidere "dove prendere i dati".

R-Type/Load/Store Datapath



Quindi:

Operazione R-Type

L'istruzione fornisce i due operandi ed il registro destinazione;

1. Si leggono i due registri
2. Un registro va direttamente all'ALU, mentre l'altro entra in un MUX
3. Viene calcolato in ALU il valore
4. Nel caso di istruzione R Type la strada è la seguente:
 1. ALU
 2. Un'istr R type non scrive o legge in memoria quindi non entra in **Write Data**

(usata invece dalle op di load store)

3. Arriva ad un altro MUX (che farà passare il segnale)
4. Il risultato viene scritto nel **Write register** (fornito dall'istruzione)

Istruzione di Load

1. Si leggono i due registri (registro su cui scrivere e registro da cui prelevare)
2. Un registro va direttamente all'ALU, mentre l'altro entra in un MUX.
Il registro 2 viene esteso (come abbiamo visto prima con l'opera d'arte del prof)
3. Viene calcolato in ALU il valore dell'indirizzo
4. Si legge in memoria dall'indirizzo appena trovato
5. Si torna dietro e si scrive nel registro

Istruzione di Store

Abbiamo un registro contenente l'indirizzo, un altro registro che contiene l'indirizzo del valore da scrivere.

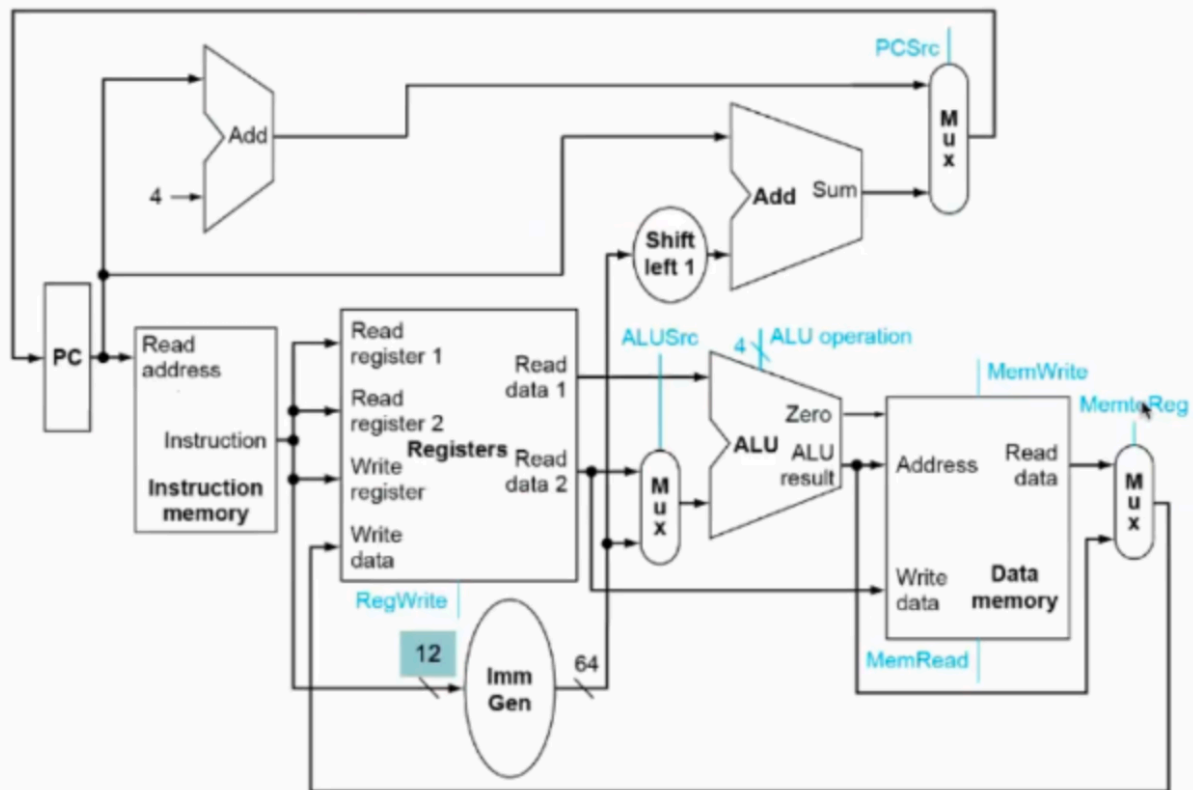
Un primo registro, come sempre, sommato ai 12 bit per trovare l'indirizzo in memoria, ovviamente sempre trovato tramite l'ALU.

Il secondo registro scavalca il primo MUX e fornisce i valori che devono essere scritti in memoria.

Non abbiamo nessun "return" in questo tipo di operazione.

Full Datapath

Full Datapath



Chapter 4 The Processor 20

Il PC viene sempre incrementato di 4, dopodiché il segnale di PC viene fatto passare attraverso un MUX; il segnale può passare senza problemi **se non si tratta di un'istr di branch**, oppure viene sommato alla costante immediata a 64 bit shiftata di 1 (che bordello speriamo che non lo chiede) e nel caso in cui il branch debba essere intrapreso (dettato dal risultato dell'output "zero" dell'ALU), il MUX sarà comandato in modo da far tornare il segnale sommato al displacement.

Controllo dell'ALU

Per controllare l'ALU vengono forniti 4 bit per decidere se l'ALU deve sommare, sottrarre, effettuare AND o OR.

Problemi di performance

Il percorso più lungo, che in gergo è detto **Critical path** è quello delle istruzioni
load: Instruction Memory -> Register file -> ALU -> data memory -> register file.

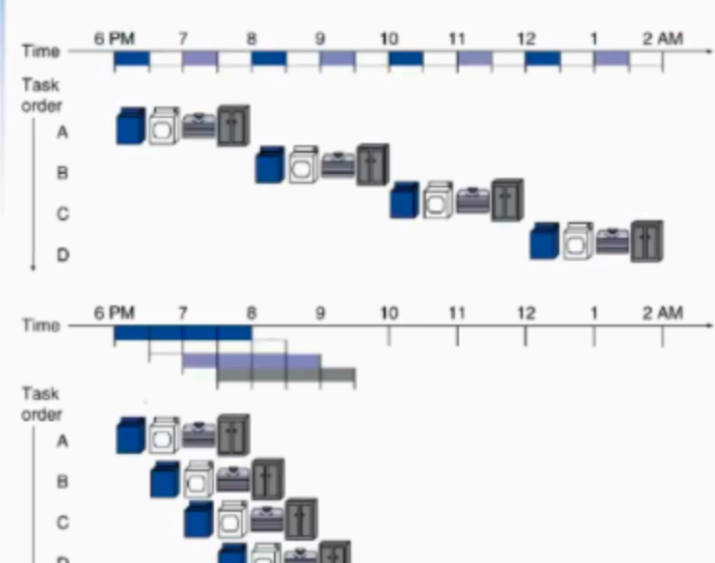
Possiamo migliorare le performance grazie alla tecnica del pipelining:

Pipelining

§4.5 An Overview of Pipelining

Pipelining Analogy

- Pipelined laundry: overlapping execution
 - Parallelism improves performance



The top chart shows four sequential tasks (A, B, C, D) each taking 3.5 hours, from 6 PM to 2 AM. The bottom chart shows the same four tasks pipelined into four stages (represented by icons of a washing machine, dryer, and iron), allowing for overlapping execution and completion by 12 PM.

- Four loads:
 - Speedup
 $= 8 / 3.5 = 2.3$
- Non-stop:
 - Speedup
 $= 2n / 0.5n + 1.5 \approx 4$
= number of stages

Chapter 4 The Processor 29

Questo esempio proviene dagli USA, e quasi in tutte le case oltre alla lavatrice è presente anche l'asciugatrice.

L'idea è quella di dover lavare asciugare e stirare i miei panni sporchi. Ho 4 carichi di lavatrice ed inizialmente si eseguono i task in sequenza lavare -> asciugare -> stendere -> stirare.

Avendo 4 carichi questa sequenza va ripetuta 4 volte, e nel primo metodo questi task vengono eseguiti in 4 alla volta, uno dopo l'altro.

Nulla vieta però di poter eseguire 4 carichi quasi simultaneamente, in modo che appena una macchina (o persona) si è liberata dal compito precedente, viene avviato il successivo.

Con questo metodo, invece di metterci (ad esempio 8 ore), ci mettiamo solo 3,5 ore, andando quindi 2.5 volte più veloce.

Oltre al fatto di metterci **totalmente** un tempo minore, abbiamo anche il vantaggio di poter avere un "prodotto finito" ogni mezz'ora.

RISC V pipeline

Abbiamo una situazione in cui dobbiamo eseguire delle istruzioni, possiamo pensare di individuare 5 stadi diversi, eseguendo un passo per stage:

1. IF: fetch
2. ID: decodifica delle istruzioni e lettura dei registri
3. EX: esecuzione delle operazioni nell'ALU
4. MEM: (non c'è sempre) accesso alla memoria
5. WB: (solo nelle op di load) la scrittura da memoria a registro

fine lezion 12