3. Evoluzione delle architetture
Indice

3. Evoluzione delle architetture
Evoluzione tecnologica

CPU
- densità di transistor: +30% per anno
- frequenza di clock: +20% per anno

Memoria
- capacità: +60% per anno
- velocità di accesso: +10% per anno
- costo per bit: -25% per anno

Disco fisso
- capacità: +60% per anno

3. Evoluzione delle architetture
Evoluzione strutturale

- L'evoluzione tecnologica corrente ha **limiti fisici** di progresso. Miglioramenti importanti necessitano di cambiamenti architetturali radicali
  - Quantum computing?
- L'aumento di prestazioni dipende sempre più da **organizzazioni** architetturali più **efficienti**
  - Individuazione e soluzione di colli di bottiglia (*bottleneck*)
  - Aumento del parallelismo interno

3. Evoluzione delle architetture
Evoluzione strutturale

- **Parallelismo**
  - Se un lavoro non può essere svolto più velocemente da una sola persona (unità), allora conviene **decomporlo** in parti che possano essere eseguite da più persone (unità) **contemporaneamente**
    - **Catena di montaggio**

3.1 Pipeline
Generalità 1

- Per svolgere un lavoro si devono eseguire tre fasi distinte e sequenziali

  L $\Rightarrow$ [*fase1*] [*fase2*] [*fase3*]
- Se ogni fase richiede un tempo **T**, un unico esecutore svolge un lavoro **L** ogni **3T**
- Per ridurre i tempi di produzione si possono utilizzare **più esecutori**

3.1 Pipeline
Generalità 2

- Soluzione (ideale) a parallelismo totale

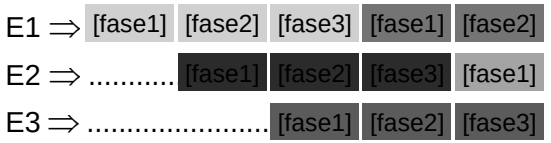
  E1 $\Rightarrow$ [fase1.**A**] [fase2.**A**] [fase3.**A**] | [fase1.**D**] …

  E2 $\Rightarrow$ [fase1.**B**] [fase2.**B**] [fase3.**B**] | [fase1.**E**] …

  E3 $\Rightarrow$ [fase1.**C**] [fase2.**C**] [fase3.**C**] | [fase1.**F**] …
- **N** esecutori svolgono un lavoro ogni **3T/N**
- Il problema è **come** preservare la **dipendenza funzionale** nell'esecuzione (di fasi) dei 'lavori' **A**, **B**, **C**, **D**, **E**, **F**, …

3.1 Pipeline
Generalità 3

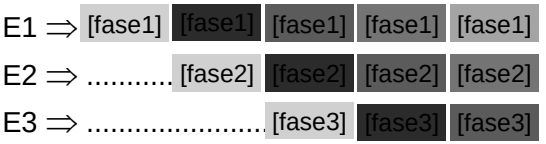
- Soluzione **pipeline** ad **esecutori generici**



- Ogni esecutore esegue un ciclo di lavoro **completo** (*sistema totalmente replicato*)
- A regime, **N** esecutori svolgono un lavoro ogni **3T/N** rispettandone la sequenza

3.1 Pipeline
Generalità 4

- Soluzione **pipeline** ad **esecutori specializzati**

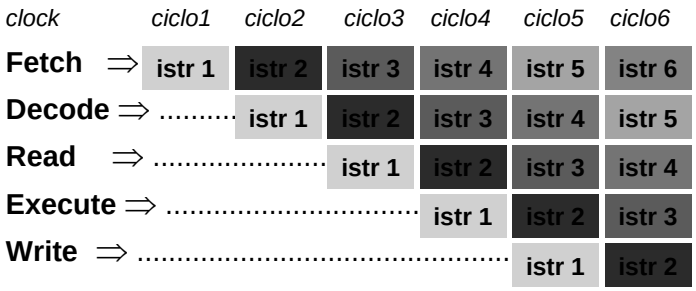


- Ogni esecutore svolge sempre e solo la **stessa** fase di lavoro
- Soluzione più efficace in termini di **uso di risorse** (**3T/N** lavori con **N/3** risorse)

3.1 Pipeline
Decomposizione in fasi

- L'esecuzione di una generica istruzione può essere suddivisa nelle seguenti fasi:
  - **fetch** lettura dell'istruzione
  - **decode** decodifica dell'istruzione
  - **read** lettura degli operandi
  - **execute** esecuzione dell'istruzione
  - **write** scrittura del risultato

3.1 Pipeline
Evoluzione ideale



3.1 Pipeline
Problemi 1

- Vari fenomeni pregiudicano il raggiungimento del massimo di parallelismo teorico (**stallo**)
  - **Sbilanciamento delle fasi**
    - Durata diversa per fase e per istruzione
  - **Problemi strutturali**
    - La sovrapposizione totale di tutte le (fasi di) istruzioni causa conflitti di accesso a risorse limitate e condivise

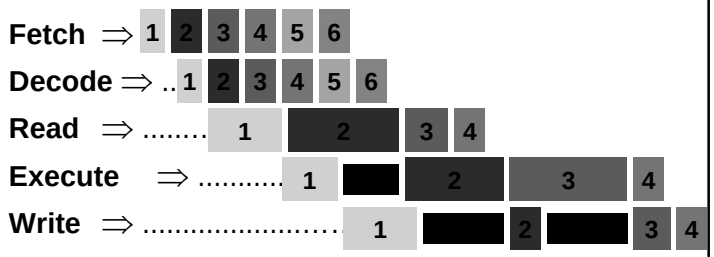
3.1 Pipeline
Problemi 2

- **Dipendenza dai dati**
  - L'operazione successiva dipende dai risultati dell'operazione precedente
- **Dipendenza dai controlli**
  - Istruzioni che causano una violazione di sequenzialità (p.es.: salti condizionali) invalidano il principio del *pipelining* sequenziale

3.1 Pipeline
Sbilanciamento delle fasi 1

- La suddivisione in fasi va fatta in base all'istruzione più onerosa
- Non tutte le istruzioni richiedono le stesse fasi e le stesse risorse
- Non tutte le fasi richiedono lo stesso tempo di esecuzione
  - P.es.: lettura di un operando tramite registro rispetto ad una mediante indirizzamento indiretto

3.1 Pipeline
Sbilanciamento delle fasi 2



Tempo di attesa forzata dovuta allo sbilanciamento delle fasi

3.1 Pipeline
Sbilanciamento delle fasi 3

Possibili soluzioni allo sbilanciamento:

- Decomporre fasi onerose in più sottofasi
  - Costo elevato e bassa utilizzazione
- Duplicare gli esecutori delle fasi più onerose e farli operare in parallelo
  - CPU moderne hanno una ALU in aritmetica intera ed una in aritmetica a virgola mobile

3.1 Pipeline
Problemi strutturali

Problemi

- Maggiori risorse interne (**severità bassa**): l'evoluzione tecnologica ha spesso permesso di duplicarle
- Colli di bottiglia (**severità alta**): l'accesso alle risorse esterne, p.es.: memoria, è molto costoso e molto frequente (anche 3 accessi per ciclo di clock)

Soluzioni

- Suddividere le memorie (accessi paralleli)
- Introdurre fasi non operative (***nop***)

3.1 Pipeline
Dipendenza dai dati 1

- Un dato modificato nella fase **execute** dell'istruzione corrente può dover essere utilizzato dalla fase **read** dell'istruzione successiva

  INC [0123]
  CMP [0123], AL

3.1 Pipeline
Dipendenza dai dati 2

Soluzioni

- Introduzione di fasi non operative (***nop***)
- Individuazione del rischio e prelievo del dato direttamente all'uscita dell'ALU (**data forwarding**)
- Risoluzione a livello di compilatore
- Riordino delle istruzioni (**pipeline scheduling**)

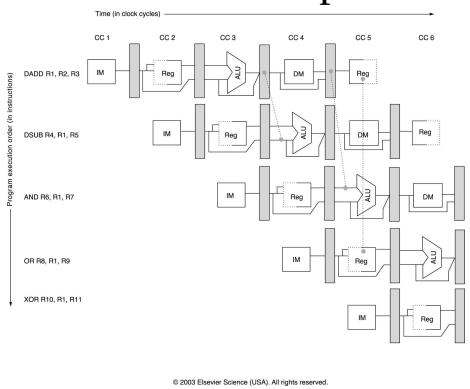
DADD R1, R2, R3
DSUB R4, R1, R5
AND R6, R1, R7
OR R8, R1, R9
XOR R10,R1,R11

clock1 clock2 clock3 clock4 clock5 clock6 clock7 clock8 clock9
F1dadd - F2dadd- F3dadd -F4dadd - F5dadd
- F1dsub- F2dsub -F3dsub - F4dsub - F5dadd
- F1and -F2and - F3and -F4and - F5and
- F1or - F2or - F3or - F4or - F5or
- F1xor - F2xor - F3xor - F4xor - F5xor

ADD R1,R2,R3
SUB R4,R1,R5

F1add - F2add- F3add -F4add - F5add
- F1sub- F2sub- F3sub - nop - F4sub

3.1 Pipeline



data forwarding

DADD R1, R2, R3
DSUB R4, R1, R5
AND R6, R1, R7
OR R8, R1, R9
XOR R10,R1,R11

3.1 Pipeline
Dipendenza dai controlli 1

- Tutte le istruzioni che modificano l'IP (salti condizionati e non, chiamate a e ritorni da procedure, interruzioni) invalidano il pipeline
- La fase **fetch** successiva carica l'istruzione seguente, che può *non essere* quella giusta
- Tali istruzioni sono circa il 30% del totale medio di un programma

3.1 Pipeline
Dipendenza dai controlli 2

Soluzioni

- Mettere in **stallo** il pipeline fino a quando non si è calcolato l'indirizzo della prossima istruzione
  - Pessima efficienza, massima semplicità
- Individuare le istruzioni critiche per anticiparne l'esecuzione, eventualmente mediante apposita logica di controllo
  - Compilazione complessa, hardware specifico

3.1 Pipeline
Dipendenza dai controlli 3

Soluzioni per salti condizionati

- Ipotizzare che non vi sia salto: se va bene siamo andati avanti, altrimenti occorre far ripartire il pipeline dall'indirizzo corretto
  - Alcune istruzioni modificano lo stato della CPU fin dalle prime fasi: nel caso bisogna ripristinare lo stato precedente (**squashing**)
- Usare due pipeline e proseguire parallelamente nelle due possibilità

3.1 Pipeline
Dipendenza dai controlli 4

Prevedere l'esito del salto

- **Staticamente**
  - Il compilatore organizza e posiziona i controlli in modo che i salti siano meno probabili
- **Dinamicamente**
  - Si costruisce una tabella storica dei salti effettuati (**Branch Target Buffer**) da usare statisticamente
    - Campi **Branch Address**, **Target Address** ed **History**

3.1 Pipeline
Super-pipeline

- Suddivisione di singole fasi di ampiezza **T** in **n** sottofasi distinte
- Attivazione di **i ≤ n** istruzioni nella medesima fase, ad intervalli regolari **T/n**
- Utilizzo di **n** pipeline operanti in parallelo

3.1 Pipeline
Pipeline *super-scalare 1*

- Esecuzione (**avvio**) di più istruzioni per ciclo di clock moltiplicando gli esecutori più onerosi e facendoli operare in parallelo
- Una o più ALU ad aritmetica intera ed una a virgola mobile

3.1 Pipeline
Pipeline *super-scalare 2*

- Solo alcuni tipi di istruzioni possono essere effettivamente avviate in parallelo
  - Istruzioni **indipendenti**, con **limitati** accessi a memoria per ciclo di clock
  - Richiede **schedulazione** statica o dinamica
- Considerevole complessità **aggiuntiva** nella logica di controllo

3.1 Pipeline
Pipeline superscalare 3

- Condizioni di Bernstein per l'indipendenza
  - Computazione **sequenziale** C_s
    $f_1 : D_1 \rightarrow R_1$ **;** $f_2 : D_2 \rightarrow R_2$
  - Computazione **parallela** C_p
    $f_1 : D_1 \rightarrow R_1$ **,** $f_2 : D_2 \rightarrow R_2$
  - $C_p \equiv C_s$ **se e solo se**:
    - $(R_1 \cap D_2 = \emptyset) \wedge (R_1 \cap R_2 = \emptyset) \wedge (R_2 \cap D_1 = \emptyset)$
  - La terza condizione diviene necessaria solo quando non si facciano ipotesi sulla durata delle operazioni e sul loro istante di inizio

3.1 Pipeline
Schedulazione dinamica 1

- La CPU **riordina dinamicamente** le istruzioni da avviare per ottimizzare l'utilizzo del pipeline, e poi **ripristina** l'ordine corretto dei risultati
- **Maggiore** è il numero di istruzioni riordinate avviate, **migliore** è l'efficienza, con aumento di complessità del control path e del data path
- **Out-of-order execution**

3.1 Pipeline
Schedulazione dinamica 2

L'avvio in parallelo di più istruzioni su più unità esecutive richiede l'uso di:

- Tabella delle annotazioni (**Scoreboard**)
  - Tabella **centralizzata** che tiene conto delle istruzioni da eseguire e delle risorse a disposizione
- Centrale di prenotazione (**Reservation station**)
  - Tecnica **distribuita** di inoltro di singole istruzioni alle varie unità, ciascuna **con coda di ingresso**

MIPS

- Architettura RISC
- Architettura molto regolare con insieme di istruzioni semplice e compatto
- Architettura progettata per una implementazione efficiente di pipeline
- Codifica delle istruzioni omogenea: 32 bit
- Co-processore per istruzioni a virgola mobile e gestione delle eccezioni

MIPS

Registri

- 32 registri di 32 bit (registro 0 contiene sempre il valore 0)
- Architettura Load / Store
  - Istruzioni di trasferimento per muovere i dati tra memoria e registri
  - Istruzioni per la manipolazione di dati operano sui valori dei registri
  - Nessuna operazione memoria ↔ memoria
- Quindi: le istruzioni operano su registri (registro i riferito con $i)
- Esempio: add $0, $1, $2

MIPS

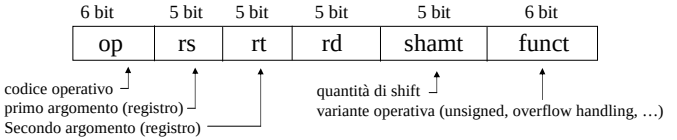
Dati e modi di indirizzamento

- Registri possono essere caricati con byte, mezze parole, e parole (riempiendo con 0 quando necessario o estendendo, cioè replicando, il segno sui bit non coinvolti del registro)
- Modalità di indirizzamento ammessi (con campi di 16 bit):
  - Immediata es. add $2, $2, 0004
  - Displacement es. sw $1, 000c($1)
- Altre modalità derivabili:
  - Indiretta (displacement a 0) es. sw $2, 0000($3)
  - Assoluta (registro 0 come registro base) es. lw $1, 00c4($0)

MIPS

Formato Istruzioni

- 32 bit per tutte, 3 formati diversi (formato R, formato I, formato J)
- **Formato R (registro)**



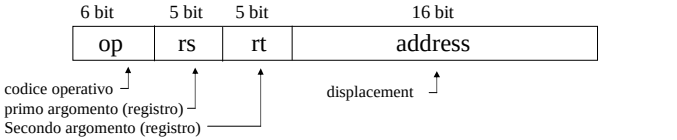
Es: `0x02484020` (**`add $8,$18,$8`** `[R8] ← [R18] + [R8]`)

op	rs	rt	rd	shamt	funct
000000	10010	01000	01000	00000	100000

MIPS

Formato Istruzioni

- **Formato I (istruzioni load / store)**



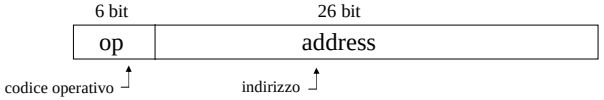
Es: `0x8D2804B0` (**`lw $8, 1200($9)`** `[R8] ← Mem[1200+[R9]]`)

op	rs	rt	address
100011	01001	01000	0000 0100 1011 0000

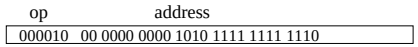
MIPS

Formato Istruzioni

- **Formato J (istruzioni jump)**



Es: **`0x0800AFFE`** (**`j 45054`** `[IP] ← 45054`)



Fasi (MIPS)

Fasi senza pipeline:

IF (istruction fetch):

- IR ← Mem[PC] ;
- NPC ← PC + 4 ;

Dove NPC è un registro temporaneo

PC (program counter) è il registro IP (instruction pointer)

Fasi (MIPS)

ID (instruction decode/register fetch cycle):

- A ← Regs[rs] ;
- B ← Regs[rt] ;
- Imm ← campo immediato di IR con segno esteso ;

Dove A, B, Imm sono registri temporanei

Fasi (MIPS)

EX (execution/effective address cycle):

1. ***Riferimento a memoria***
   - ALUOutput ← A + Imm ;
2. ***Istruzione ALU registro-registro***
   - ALUOutput ← A ***func*** B ;
3. ***Istruzione ALU registro-immediato***
   - ALUOutput ← A ***op*** Imm ;
4. ***Salto***
   - ALUOutput ← NPC + (Imm << 2);
   - Cond ← (A = = 0) ;

Fasi (MIPS)

MEM (memory access/branch completion cycle):

- PC ← NPC ; in tutti i casi

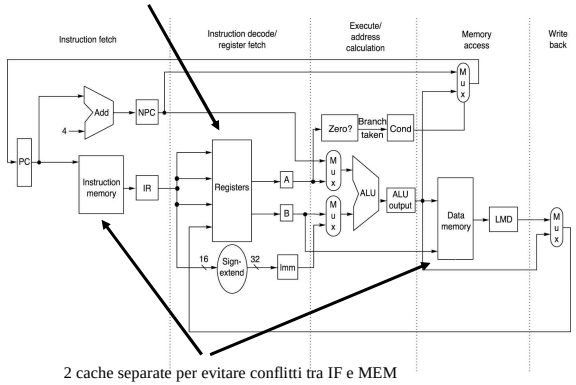
1. ***Riferimento a memoria***
   - LMD ← Mem[ALUOutput] or Mem[ALUOutput] ← B;
2. ***Salto***
   - if (cond) PC ← ALUOutput ;

Fasi (MIPS)

WB (write/back cycle):

1. ***Istruzione ALU registro-registro***
   - Regs[rd] ← ALUOutput ;
2. ***Istruzione ALU registro-immediato***
   - Regs[rt] ← ALUOutput ;
3. ***Istruzione Load***
   - Regs[rt] ← LMD ;

registri letti (anche 2 volte) e scritti nello stesso ciclo di clock per evitare conflitti fra IS e WB



2 cache separate per evitare conflitti tra IF e MEM

Pipeline (MIPS)

- Architettura che si presta ad una facile introduzione della pipeline: uno stadio per fase, i ciclo di clock per stadio
- Occorre memorizzare i dati fra una fase e la successiva: si introducono opportuni registri (denominati pipeline registers o pipeline latches) fra i vari stadi della pipeline
- Tali registri memorizzano sia dati che segnali di controllo che devono transitare da uno stadio al successivo
- Dati che servono a stadi non immediatamente successivi vengono comunque copiati nei registri dello stato successivo per garantire la correttezza dei dati

Pipeline (MIPS)