Richiami sull'architettura del processore MIPS a 32 bit

Architetture Avanzate dei Calcolatori

Valeria Cardellini

Caratteristiche principali dell'architettura del processore MIPS

- E' un'architettura RISC (Reduced Instruction Set Computer)
  - Eseguire soltanto istruzioni brevi in un ciclo base ridotto, con l'obiettivo di migliorare le prestazioni fornite dalle architetture CICS
    - Principi progettuali di semplicità e regolarità
    - Rendere il caso più comune il più veloce possibile
- Architettura di tipo registro-registro (load/store)
  - Gli operandi della ALU possono provenire solo dai registri di uso generale interni al processore e non possono provenire direttamente dalla memoria
    - Operazioni di *load*: caricamento dei dati dalla memoria ai registri del processore
    - Operazioni di *store*: memorizzazione dei dati contenuti nei registri del processore in memoria

Richiami riguardanti …

- Insieme di istruzioni semplificato
- Esecuzione delle istruzioni
- Struttura del processore

Insieme di istruzioni semplificato del MIPS

- Consideriamo un'implementazione semplificata del processore MIPS
  - In particolare, no istruzioni di I/O
- Tre classi di istruzioni
  - Istruzioni logico-aritmetiche
  - Istruzioni di trasferimento da/verso la memoria (load/store)
  - Istruzioni di salto (condizionato e incondizionato) per il controllo del flusso di programma

Esempi di istruzioni

- Istruzioni logico-aritmetiche
  - add $s0, $s0, $s1 # $s0 = $s0 + $s1
  - addi $t0, $t0, 1 # $t0 = $t0 + 1
  - and $s1, $s2, $s3 # $s1 = $s2 && $s3
- Istruzioni di trasferimento da/verso la memoria
  - lw $s1, 8($s2) # $s1 = M[$s2+8]
  - sw $s1, 8($s2) # M[$s2+8] = $s1

Esempi di istruzioni (2)

- Istruzioni di salto condizionato
  - beq $s0, $s1, L1 # salta a L1 se ($s0 == $s1)
  - bne $s0, $s1, L1 # salta a L1 se ($s0 != $s1)
- Istruzioni di salto incondizionato
  - j L1 # salta a L1
  - jr $s1 # salta all'indirizzo in $s1
  - jal L1 # salta a L1; salva l'indirizzo della prossima istruzione in $ra

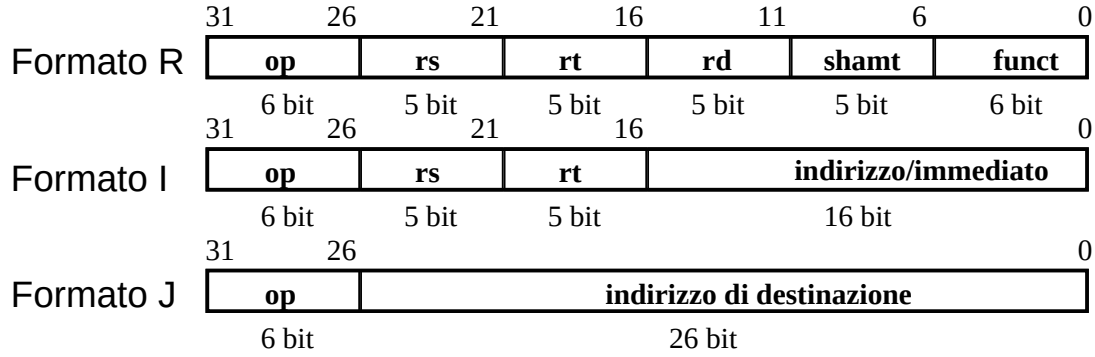
Registri del processore MIPS a 32 bit

- 32 registri di uso generale
  - Per convenzione si usano nomi simbolici preceduti da $
    - $s0, $s1, …, $s7 (detti registri saved) per contenere variabili
    - $t0, $t1, …, $t9 (detti registri temporanei) per contenere variabili temporanee
  - I registri possono anche essere indicati dal loro numero preceduto da $: $0, …, $31
- Dei 32 registri alcuni sono special-purpose, ossia dedicati per l'esecuzione di alcune istruzioni
  - Es.: $ra (= $31) è il registro di ritorno
- 32 registri in virgola mobile
  - $f0, …, $f31

Formato delle istruzioni MIPS

- Tutte le istruzioni MIPS hanno la stessa lunghezza (32 bit)
- Le istruzioni MIPS sono di 3 formati
  - Formato R (registro)
    - Istruzioni logico-aritmetiche
  - Formato I (immediato)
    - Istruzioni con operandi immediati
    - Istruzioni di accesso in memoria (load/store)
    - Istruzioni di salto condizionato
  - Formato J (jump)
    - Istruzioni di salto incondizionato
- I diversi formati sono riconosciuti tramite il valore dei 6 bit del primo campo, detto codice operativo (*opcode*), che indica come trattare i rimanenti 26 bit dell'istruzione

Formato delle istruzioni MIPS (2)



- I campi delle istruzioni
  - *op* (opcode): identifica il formato di istruzione
  - *rs*, *rt*, *rd*: registri sorgente (*rs* e *rt*) e registro destinazione (*rd*)
  - *shamt* (shift amount): scorrimento
  - *funct*: indica la variante specifica dell'operazione nel campo *op*
  - *indirizzo/immediato*: offset dell'indirizzo o valore immediato
  - *indirizzo di destinazione*: una parte dell'indirizzo assoluto di destinazione del salto incondizionato

Modalità di indirizzamento

- Le modalità di indirizzamento indicano i diversi modi con i quali fare riferimento agli operandi nelle istruzioni
- MIPS ha solo 5 modalità di indirizzamento
  - Tramite registro
  - Immediato
  - Tramite base o spiazzamento
  - Relativo al Program Counter
  - Pseudo-diretto
- Una singola istruzione può usare più modalità di indirizzamento (es.: addi $t0, $t0, 4)

Modalità di indirizzamento (2)

- Indirizzamento tramite registro
  - L'operando è il contenuto di un registro del processore
  - Es: add $s0, $s1, $s2 (formato R)
- Indirizzamento immediato
  - L'operando è una costante, il cui valore è specificato nell'istruzione
  - Es: addi $s0, $s1, 1 (formato I)
- Indirizzamento con base o spiazzamento
  - L'operando è in una locazione di memoria, il cui indirizzo si ottiene sommando il contenuto di un registro base ad un valore costante (*offset* o spiazzamento) specificato nell'istruzione
  - Es: lw $t1, 4($s0) (formato I)

Modalità di indirizzamento (3)

- Indirizzamento relativo al Program Counter
  - L'operando è in una locazione di memoria, il cui indirizzo si ottiene sommando il contenuto del Program Counter (PC) ad un valore costante (*offset* o spiazzamento) specificato nell'istruzione
  - Es: beq $s0, $s1, L1 (formato I)
- Indirizzamento pseudo-diretto
  - Una parte dell'indirizzo è presente come valore costante (offset) nell'istruzione, ma deve essere completato
  - L'indirizzo di destinazione del salto si ottiene traslando a sinistra di 2 bit i 26 bit di offset specificati nell'istruzione e concatenando i 28 bit così ottenuti con i 4 bit più significativi del PC
  - Es: j L2 (formato J)

Il processore

- Distinguiamo unità di elaborazione e unità di controllo
- Unità di elaborazione dati
  - Hardware per compiere le operazioni necessarie all'esecuzione delle istruzioni
- Unità di controllo
  - Riceve dei segnali di ingresso e genera in uscita segnali per la lettura/scrittura degli elementi di memoria, segnali di selezione per i multiplexer, segnali per il controllo della ALU
- Implementazione dell'unità di elaborazione a ciclo singolo
  - Tutte le istruzioni vengono eseguite in un solo ciclo di clock, la cui lunghezza è determinata dal percorso critico
  - **E' l'implementazione usata per il pipelining**
- Implementazione dell'unità di elaborazione a ciclo multiplo
  - L'esecuzione di un'istruzione richiede più cicli di clock

I cinque passi delle istruzioni

- I cinque passi di esecuzione delle istruzioni
  - Prelievo dell'istruzione (Instruction fetch: **IF**)
  - Decodifica dell'istruzione/caricamento dei registri (Instruction decode: **ID**)
  - Esecuzione/calcolo dell'indirizzo di memoria (Execute: **EX**)
  - Accesso alla memoria (Memory access: **MEM**)
  - Scrittura del risultato (Write-back: **WB**)

IF Instruction Fetch	ID Instruction Decode	EX EXecute	MEM MEMory access	WB Write-Back

Esecuzione delle istruzioni logico-aritmetiche

- Esempio: op $x, $y, $z
- Un'istruzione logico-aritmetica viene eseguita in 4 passi nell'implementazione a ciclo singolo
  1. Prelievo dell'istruzione dalla memoria istruzioni e incremento del PC
  2. Lettura dei due registri sorgente ($y e $z) dal banco dei registri
  3. Esecuzione dell'operazione (op) da parte dell'ALU sui valori letti dal banco dei registri
  4. Scrittura del risultato dell'ALU nel registro destinazione ($x) del banco dei registri

Esecuzione delle istruzioni di load

- Esempio: lw $x, offset($y)
- Un'istruzione di load viene eseguita in 5 passi nell'implementazione a ciclo singolo
  1. Prelievo dell'istruzione dalla memoria istruzioni e incremento del PC
  2. Lettura del registro base ($y) dal banco dei registri
  3. Esecuzione dell'operazione (somma) da parte dell'ALU per calcolare l'indirizzo di memoria ($y + offset)
  4. Lettura del dato dalla memoria dati (M[$y + offset]) utilizzando come indirizzo il risultato della ALU
  5. Scrittura del dato proveniente dalla memoria nel registro destinazione ($x) del banco dei registri

Esecuzione delle istruzioni di store

- Esempio: sw $x, offset($y)
- Un'istruzione di store viene eseguita in 4 passi nell'implementazione a ciclo singolo
  1. Prelievo dell'istruzione dalla memoria istruzioni e incremento del PC
  2. Lettura del registro base ($y) dal banco dei registri
  3. Esecuzione dell'operazione (somma) da parte dell'ALU per calcolare l'indirizzo di memoria ($y + offset)
  4. Scrittura del dato proveniente dal banco dei registri ($x) nella memoria dati (M[$y + offset]) utilizzando come indirizzo il risultato della ALU

Esecuzione delle istruzioni di salto condizionato

- Esempio: beq $x, $y, offset
- Un'istruzione di salto condizionato viene eseguita in 4 passi nell'implementazione a ciclo singolo
  1. Prelievo dell'istruzione dalla memoria istruzioni e incremento del PC
  2. Lettura dei due registri sorgente ($x e $y) dal banco dei registri
  3. Esecuzione dell'operazione (sottrazione) da parte dell'ALU per confrontare i valori letti dal banco dei registri ($x - $y) e calcolo dell'indirizzo di destinazione del salto
  4. L'uscita Zero della ALU viene utilizzata per decidere quale valore deve essere memorizzato nel PC: (PC+4) oppure (PC+4+offset)

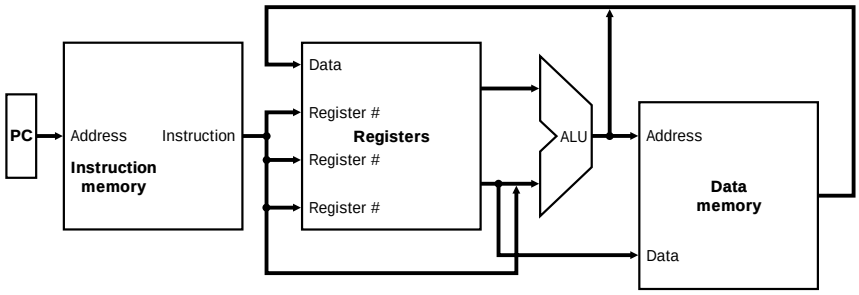
Esecuzione delle istruzioni

- Per ogni tipo di istruzione, i primi due passi da eseguire sono uguali
  - Prelievo dell'istruzione dalla memoria istruzioni e incremento del PC
  - Lettura di uno o due registri dal banco dei registri, selezionando i registri a cui accedere tramite i campi dell'istruzione
- Le azioni successive dipendono dal tipo di istruzione (codice operativo), sebbene tutte le istruzioni utilizzino l'ALU dopo il secondo passo
  - Le istruzioni logico-aritmetiche per eseguire l'operazione
  - Le istruzioni di load e store per calcolare l'indirizzo di memoria
  - Le istruzioni di salto condizionato per effettuare il confronto

Esecuzione delle istruzioni (2)

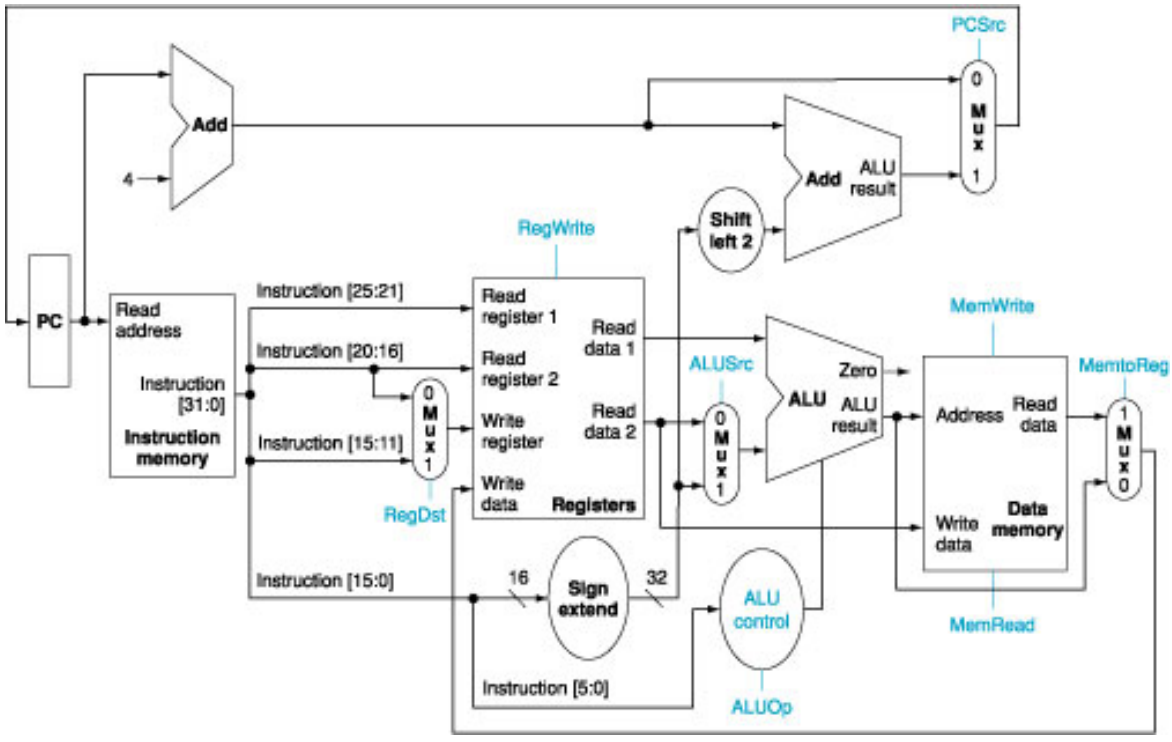
- Dopo aver utilizzato l'ALU, le azioni richieste per completare le varie istruzioni si differenziano ulteriormente
  - Le istruzioni logico-aritmetiche devono scrivere il risultato della ALU nel registro destinazione
  - Le istruzioni di load richiedono l'accesso in lettura alla memoria dati ed eseguono il caricamento del dato letto nel registro di destinazione
  - Le istruzioni di store richiedono l'accesso in scrittura alla memoria dati ed eseguono la memorizzazione del dato proveniente dal registro sorgente
  - Le istruzioni di salto condizionato possono scrivere il valore del PC in base al risultato del confronto

Struttura di base del processore MIPS



- Per eseguire tutte le istruzioni in un solo ciclo di clock
  - Ogni risorsa/unità funzionale può essere utilizzata una sola volta per istruzione
  - Occorre duplicare le risorse/unità funzionali di cui si ha bisogno più di una volta nello stesso ciclo di clock
    - Memoria dati distinta dalla memoria istruzioni
    - ALU e sommatori
  - Alcune risorse/unità funzionali possono essere condivise da differenti flussi di esecuzione
    - Tramite l'introduzione di multiplexer

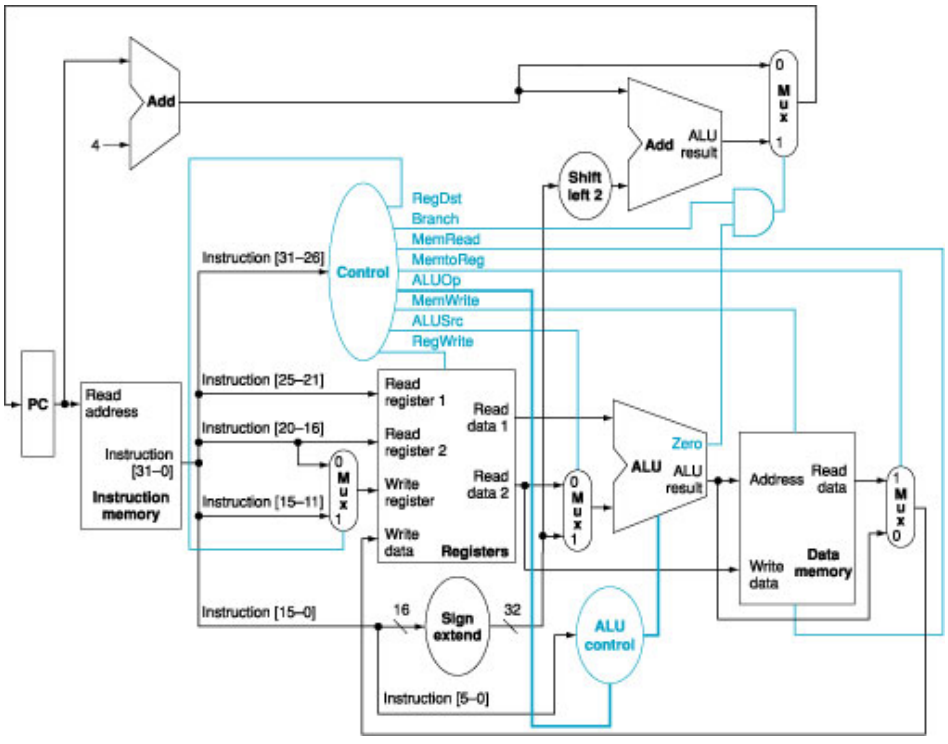
L'unità di elaborazione a ciclo singolo con i segnali di controllo



Il significato dei segnali di controllo

Segnale	Effetto quando vale 0	Effetto quando vale 1
RegDst	Registro destinazione = rt	Registro destinazione = rd
RegWrite	Nessuno	Nel registro indicato sull'ingresso *Write register* viene scritto il valore *Write data*
ALUSrc	Il secondo operando di ALU viene da *Read data 2*	Il secondo operando di ALU viene dall'estensione di segno
PCSrc	Scrittura di PC con PC+4	Scrittura di PC con l'output del sommatore per il branch
MemRead	Nessuno	Lettura della locazione di memoria indicata da *Address*
MemWrite	Nessuno	Scrittura della locazione di memoria indicata da *Address*
MemtoReg	L'ingresso *Write data* (banco registri) viene dalla ALU	L'ingresso *Write data* (banco registri) viene dalla memoria dati

L'unità di elaborazione a ciclo singolo con l'unità di controllo



Il valore dei segnali di controllo

Istruzione	RegDst	ALUSrc	Memto Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp
tipo-R	1	0	0	1	0	0	0	10
lw	0	1	1	1	1	0	0	00
sw	X	1	X	0	0	1	0	00
beq	X	0	X	0	0	0	1	01

L'unità di elaborazione a ciclo multiplo

- L'esecuzione di un'istruzione è distribuita su più cicli
- Si utilizza un ciclo di clock di durata inferiore rispetto a quello dell'implementazione a ciclo singolo
- Implementazione dell'unità di elaborazione a ciclo multiplo
  - Più complessa del ciclo singolo
  - Ogni fase di esecuzione di un'istruzione richiede un ciclo di clock
  - Un'unità funzionale può essere usata più di una volta per istruzione in cicli differenti (condivisione di unità funzionali ed eliminazione di ridondanze hardware)
    - Singola unità di memoria per istruzioni e dati anziché due memorie distinte
    - Una sola ALU anziché una ALU e due sommatori
  - Occorre introdurre dei registri interni addizionali per memorizzare i valori da usare nei cicli di clock successivi