


Architettura degli elaboratori - II
Introduzione

Prof. Alberto Borghese
Dipartimento di Scienze dell'Informazione
borghese@dsi.unimi.it

Università degli Studi di Milano

Riferimento sul Patterson: capitolo 4.2 , 4.4, D1, D2.


Introduzione alla CPU

- Introduzione
- CPU

Architetture II (6cfu)

Docente: Prof. N. Alberto Borghese: borghese@dsi.unimi.it
Laboratorio Assembler:
Dott. Iuri Frosio: frosio@dsi.unimi.it
Dott. Massimo Marchi: marchi@dsi.unimi.it

Orario e aule:

Lunedì	Ore 8.30-10.30	Aula 201, Via Celoria 20
Giovedì	Ore 8.30-10.30	Aula 201, Via Celoria 20
Venerdì	**Ore 10.30-12.30**	**Aula V307, Via Celoria 20 (Cognomi A-F)**
Venerdì	**Ore 10.30-12.30**	**Aula V309, Via Celoria 20 (Cognomi G-Z)**

Orario di ricevimento: su appuntamento.

Strumento principale di contatto: email!

Programma

Sito principale:
http://homes.dsi.unimi.it/~borghese/Teaching/Architettura_II/_Arch_II.html

Programma:
http://homes.dsi.unimi.it/~borghese/Teaching/Architettura_II/Programma_2010-2011.html

Argomenti principali:
- CPU (avanzate)
- Gerarchie di memoria
- Interconnessioni

Esame

Parte teorica (2/3 del voto).

- Prova scritta + orale. Appelli ogni 1 / 2 / 3 mesi, al di fuori dal periodo delle lezioni.
- 2 compitini in itinere durante l'anno. I compitini sostituiscono interamente scritto e orale.
- Per superare la parte di teoria con i compitini occorre avere preso almeno 17 in tutti i compitini e che la media dei 3 compitini sia ≥ 18. I compitini sono consigliati solo a chi frequenta.
- L'orale con i compitini è facoltativo.

Progetto di laboratorio in Assembler (PC-Spim, 1/3 del voto).

Materiale didattico

See web page

http://homes.dsi.unimi.it/~borghese/Teaching/Architettura_II/References.rtf

Testo di base (è disponibile sia in inglese che in italiano):
Struttura e progetto dei calcolatori: l'interfaccia hardware-software, D.A. Patterson and J.L. Hennessy, Terza edizione, Zanichelli, estate 2010 (Nota: la terza edizione Zanichelli è la traduzione della quarta edizione inglese).
"Computer Organization & Design: The Hardware/Software Interface", D.A. Patterson and J.L. Hennessy, Morgan Kaufmann Publishers, Fourth Edition, 2009.
Potete trovare esercizi del testo svolti al seguente URL:
http://books.elsevier.com/companions/1558606041/.

Obbiettivo di un'architettura

Elabora in modo adeguato un input per produrre l'output.

- Le unità di *ingresso* (tastiera, mouse, rete, interfacce con dispositivi di acquisizione, ecc.) permettono al calcolatore di acquisire informazioni dall'ambiente esterno.
- L'architettura di elaborazione.

- Le unità di *uscita* (terminale grafico, stampanti, rete, ecc.) consentono al calcolatore di comunicare i risultati ottenuti dall'elaborazione all'ambiente esterno.

Cosa fa un elaboratore?

- Algoritmi (sequenza di istruzioni).
  Calcoli (calcolatore).
  Operazioni logiche (elaboratore).
- Programma (Ada Lovelace, 1830) = *Algoritmi in Software.*

Come lo fa? *Hardware*.

Input ==> Elaborazione ==> Output

- Terza rivoluzione della nostra civiltà: la rivoluzione agricola, la rivoluzione industriale e la rivoluzione dell'informatica.

CPU di tipo CISC (*Complex Instruction Set Computer*)

- Caratterizzate da elevata complessità delle istruzioni eseguibili ed elevato numero di istruzioni che costituiscono l'insieme delle istruzioni.
- Numerose modalità di indirizzamento per gli **operandi** dell'*ALU* che possono provenire da registri oppure da memoria, nel qual caso l'indirizzamento può essere diretto, indiretto, con registro base, ecc.
- Dimensione *variabile* delle istruzioni a seconda della modalità di indirizzamento di ogni operando ⇒ complessità di gestione della fase di prelievo o *fetch* in quanto a priori non è nota la lunghezza dell'istruzione da caricare.
- Elevata complessità della *CPU* stessa (cioè dell'hardware relativo) in termini degli elementi che la compongono con la conseguenza di rallentare i tempi di esecuzione delle operazioni. Elevata profondità dell'albero delle porte logiche, utilizzato per la decodifica.

Utilizzo architettura Intel 80x86: le 10 istruzioni più frequenti

° Rank	instruction	Integer Average Percent total executed
1	load	22%
2	conditional branch	20%
3	compare	16%
4	store	12%
5	add	8%
6	and	6%
7	sub	5%
8	move register-register	4%
9	call	1%
10	return	1%
	Total	96%

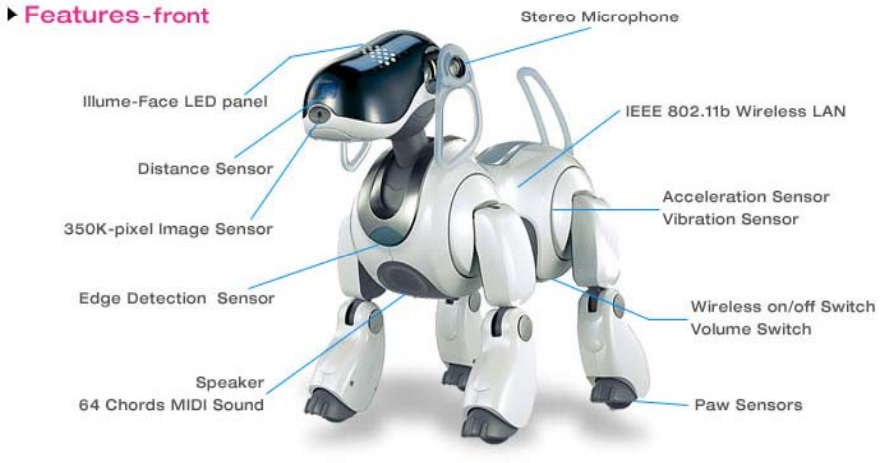
° **Simple instructions dominate instruction frequency** => RISC

Architetture di tipo RISC (*Reduced Instruction Set Computer*)

- Ispirate al principio di eseguire soltanto istruzioni semplici: le operazioni complesse vengono scomposte in una serie di istruzioni più semplici da eseguire in un ciclo base ridotto, con l'obiettivo di migliorare le prestazioni ottenibili dalle *CPU CISC*.
- Caratterizzate da istruzioni molto semplificate.
- Gli operandi dell'*ALU* possono provenire dai registri ma *non* dalla memoria. Per il trasferimento dei dati da memoria ai registri e viceversa si utilizzano delle apposite operazioni di caricamento (*load*) e di memorizzazione (*store)*
  ⇒ *architetture load/store*.
- *CPU* relativamente semplice ⇒ si riducono i tempi di esecuzione delle singole istruzioni, che sono però meno potenti delle istruzioni *CISC*.
- Dimensione *fissa* delle istruzioni ⇒ più semplice la gestione della fase di prelievo *(fetch)* e della codifica delle istruzioni da eseguire

Architettura base del corso – MIPS MIPS Technologies

AIBO (Sony, 2003) – MIPS 7000, sistemi embedded che montano Windows CE, PlayStation 2, router, gateway...



Architettura MIPS

- Architettura MIPS appartiene alla famiglia delle architetture **RISC (Reduced Instruction Set Computer)** sviluppate dal 1980 in poi
  - Esempi: Sun Sparc, HP PA-RISC, IBM Power PC, DEC Alpha, Silicon Graphics, AIBO-Sony, ARM.
- Principali obiettivi delle architetture RISC:
  - Semplificare la progettazione dell'hardware e del compilatore
  - Massimizzare le prestazioni
  - Minimizzare i costi

Contenuto di un'istruzione

Tutte le istruzioni MIPS hanno la **stessa** dimensione **(32 bit) – Architettura RISC.**

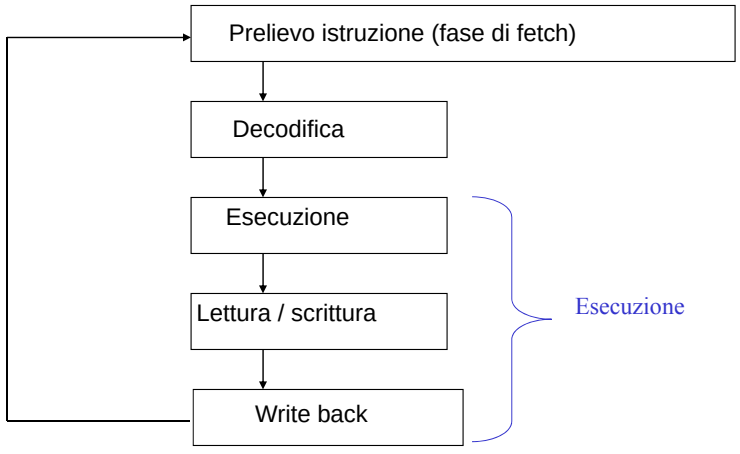
Alcune domande:
- Come e dove si specifica il tipo di istruzione?
- Come e dove si specifica da dove vengono letti i dati?
- Come e dove si specifica dove si scrivono i dati prodotti?
- Come viene gestita la memoria in lettura e scrittura?
- Come vengono gestiti i salti?

Codifica delle istruzioni

- I 32 bit hanno un significato diverso a seconda del formato (o tipo) di istruzione
  - il tipo di istruzione è riconosciuto in base al valore di alcuni bit **(6 bit)** più significativi **(codice operativo - *OPCODE*)**
- Le istruzioni MIPS sono di **3** tipi (formati):
  - **Tipo R (register) – Lavorano su 3 registri.**
    - Istruzioni aritmetico-logiche.
  - **Tipo I (immediate) – Lavorano su 2 registri. L'istruzione è suddivisa in un gruppo di 16 bit contenenti informazioni + 16 bit riservati ad una costante.**
    - Istruzioni di accesso alla memoria o operazioni contenenti delle costanti.
  - **Tipo J (jump) – Lavora senza registri: codice operativo + indirizzo di salto.**
    - Istruzioni di salto incondizionato.

	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
R	op	rs	rt	rd	shamt	funct
I	op	rs	rt	indirizzo		
J	op	indirizzo				

Ciclo di esecuzione di un'istruzione MIPS



Lettura dell'istruzione (fetch)

- Istruzioni e dati risiedono nella memoria principale, dove sono stati caricati attraverso un'unità di ingresso.
- L'esecuzione di un programma inizia quando il registro PC punta alla (contiene l'indirizzo della) prima istruzione del programma in memoria.
- Il segnale di controllo per la lettura (READ) viene inviato alla memoria.
- Trascorso il tempo necessario all'accesso in memoria, la parola indirizzata (in questo caso la prima istruzione del programma) viene letta dalla memoria e trasferita nel registro IR.
- Il contenuto del PC viene incrementato in modo da puntare all'istruzione successiva.

Decodifica dell'istruzione

- L'istruzione contenuta nel registro IR viene decodificata per essere eseguita. Alla fase di decodifica corrisponde la predisposizione della CPU (apertura delle vie dicomunicazione appropriate) all'esecuzione dell'istruzione.
- In questa fase vengono anche recuperati gli operandi. Nelle architetture MIPS gli operandi possono essere solamente nel Register File oppure letti dalla memoria.

.

  - Architetture a registri:
    - Se un operando risiede in memoria, deve essere prelevato caricando l'indirizzo dell'operando nel registro MAR della memoria e attivando un ciclo di READ della memoria.
    - L'operando letto dalla memoria viene posto nel registro della memoria MDR per essere trasferito alla ALU, che esegue l'operazione. Nelle architetture MIPS, l'operando viene trasferito nel Register file nella fase di Scrittura.
  - Architetture LOAD/STORE:
    - Le istruzioni di caricamente dalla memoria sono separate da quelle aritmetico/logiche.

Calcolo dell'istruzione (esecuzione)

Viene selezionato il circuito / i circuiti combinatori appropriati per l'esecuzione delle operazioni previste dall'istruzione e determinate in fase di decodifica.

Tra le operazioni previste, c'è anche la formazione dell'indirizzo di memoria da cui leggere o su cui scrivere un dato.

Lettura / Scrittura in memoria

In questa fase il dato presente in un registro, viene scritto in memoria oppure viene letto dalla memoria un dato e trasferito ad un registro.

Questa fase non è richiesta da tutte le istruzioni

Nel caso particolare di Architetture LOAD/STORE, quali MIPS, le istruzioni di caricamente dalla memoria sono separate da quelle aritmetico/logiche. Se effettuo una Letura / Scrittura, **non** eseguo operazioni aritmetico logiche sui dati.

Sistema di memoria "sganciato" dalla coppia register-file + CPU.

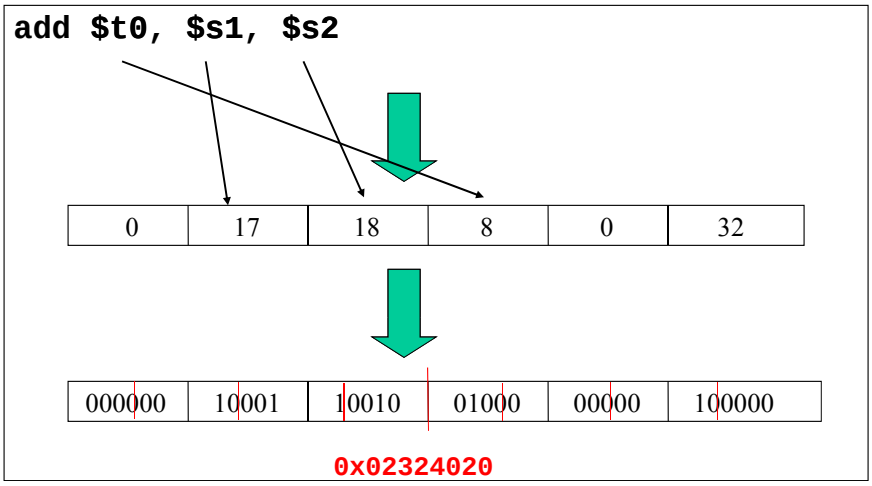
Scrittura in register file (write-back)

- Il risultato dell'operazione può essere memorizzato nei registri ad uso generale oppure in memoria.
- Non appena è terminato il ciclo di esecuzione dell'istruzione corrente (termina la fase di Write Back), si preleva l'istruzione successiva dalla memoria.

MIPS: Software conventions for Registers

0	zero	constant 0
1	at	reserved for assembler
2	v0	expression evaluation &
3	v1	function results
4	a0	arguments
5	a1	
6	a2	
7	a3	
8	t0	temporary: caller saves
. . .		(callee can clobber)
15	t7	
16	s0	callee saves
. . .		(caller can clobber)
23	s7	
24	t8	temporary (cont'd)
25	t9	
26	k0	reserved for OS kernel
27	k1	
28	gp	Pointer to global area
29	sp	Stack pointer
30	fp	frame pointer (s8)
31	ra	Return Address (HW)

Istruzioni di tipo R: esempio



Istruzioni di tipo R: esempi

Nome campo	op	rs	rt	rd	shamt	funct
Dimensione	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
`sub $s1, $s2, $s3`	000000	10010	10011	10001	00000	100010

Nome campo	op	rs	rt	rd	shamt	funct
Dimensione	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
`and $s1, $s2, $s3`	000000	10010	10011	10001	00000	100100

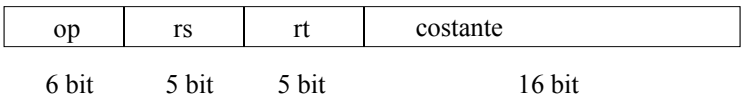
Nome campo	op	rs	rt	rd	shamt	funct
Dimensione	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
`sll $s1, $s2, 7`	000000	X	10010	10001	00111 (7)	000000

$s1 = s2*2^7$

Nome campo	op	rs	rt	rd	shamt	funct
Dimensione	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
`srl $s1, $s2, 7`	000000	X	10010	10001	00111 (7)	000010

$s1 = s2*2^{-7}$

Formato istruzioni di tipo I



- In questo caso, i campi hanno il seguente significato:
  - **`op`** identifica il tipo di istruzione;
  - `rs` indica il registro sorgente. Nel caso di una lw contiene il registro base;
  - **`rt`** indica il registro target. Nel caso di una lw, contiene il registro destinazione dell'istruzione di caricamento;
  - **`costante`**. Nel caso di una lw riporta lo spiazzamento (offset).

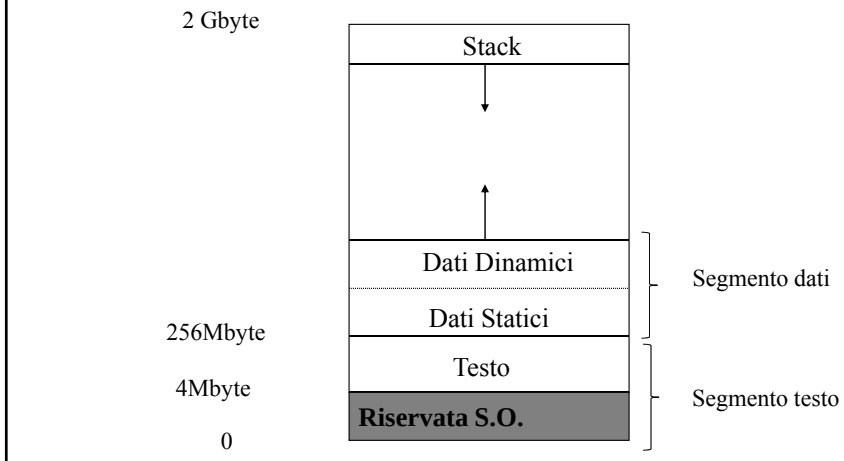
Versione I di istruzioni aritmetico-logiche

Nome campo	op	rs	rt	costante
Dimensione	6-bit	5-bit	5-bit	16-bit
`addi $s1, $s2, 4`	001000	10001	10001	0000 0000 0000 0100

Nome campo	op	rs	rt	costante
Dimensione	6-bit	5-bit	5-bit	16-bit
`slti $t0, $s2, 8`	001010	10010	01000	0000 0000 0000 1000

```
# $t0 = 1 if $s2 < 8
```

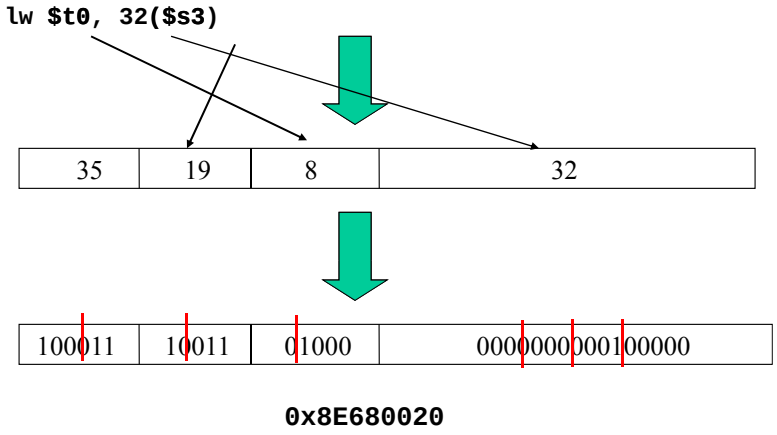
Organizzazione logica della memoria



Indirizzamento della memoria dati

Istruzioni di tipo I: esempio

Con questo formato una istruzione **lw (sw)** può indirizzare byte nell'intervallo **-2^{15}** (-32K) ÷ **$+2^{15}-1$** (32K -1) rispetto all'indirizzo base: indirizzo = indirizzo_base + offset.



Istruzioni di tipo I: esempi

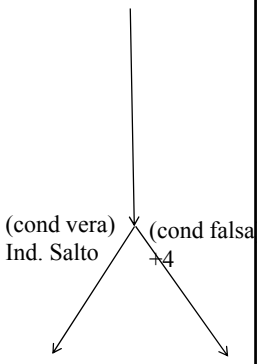
Nome campo	op	rs	rt	indirizzo
Dimensione	6-bit	5-bit	5-bit	16-bit
`lw $t0, 32 ($s3)`	100011	10011	01000	0000 0000 0010 0000

Nome campo	op	rs	rt	indirizzo
Dimensione	6-bit	5-bit	5-bit	16-bit
`sw $t0, 32 ($s3)`	101011	10011	01000	0000 0000 0010 0000

Nome campo	op	rs	rt	indirizzo
Dimensione	6-bit	5-bit	5-bit	16-bit
`addi $t0, $s3, 64`	001000	10011	01000	0000 0000 0100 0000

Istruzioni di salto condizionato

- Salti condizionati relativi:
  - **`beq r1, r2, L1`** (*branch on equal*)
  - **`bne r1, r2, L1`** (*branch on not equal*)
- Salti condizionati relativi:
  - Il flusso sequenziale di controllo cambia solo se la condizione è vera. (beq)
  - Il calcolo del valore dell'etichetta **`L1`** (indirizzo di destinazione del salto) è relativo al Program Counter (PC).



Branch: esempi

Nome campo	op	rs	rt	indirizzo
Dimensione	6-bit	5-bit	5-bit	16-bit
`beq $s1, $s2, 100`	000100	10001	10010	0000 0000 0001 1001

L1 = 100 in byte Codifica su 18 bit: (00) 000 0000 0001 1001(00) in binario.

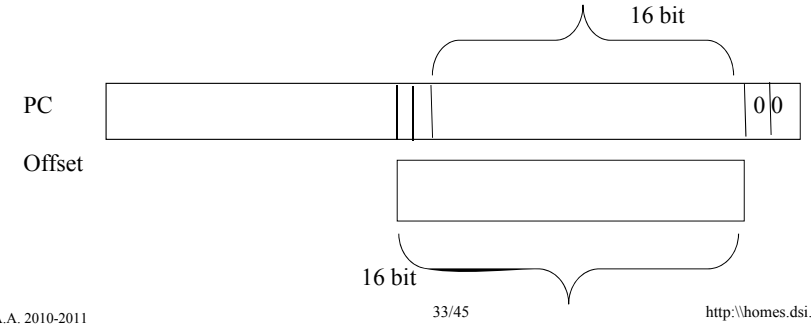
Nome campo	op	rs	rt	indirizzo
Dimensione	6-bit	5-bit	5-bit	16-bit
`beq $s1, $s2, -100`	000100	10001	10010	1111 1111 1110 0111

L1 = -100 in byte Codifica su 18 bit: (11)111 1111 1110 0111(00) in binario.

Allargamento dello spazio di indirizzamento

0000	0	0
0100	1	4
1000	2	8
1100	3	12

Considero 64Mword invece di 64Kbyte. Lo spazio indirizzabile all'interno del segmento di testo è di 64Kword * 4 = 256Kbyte.



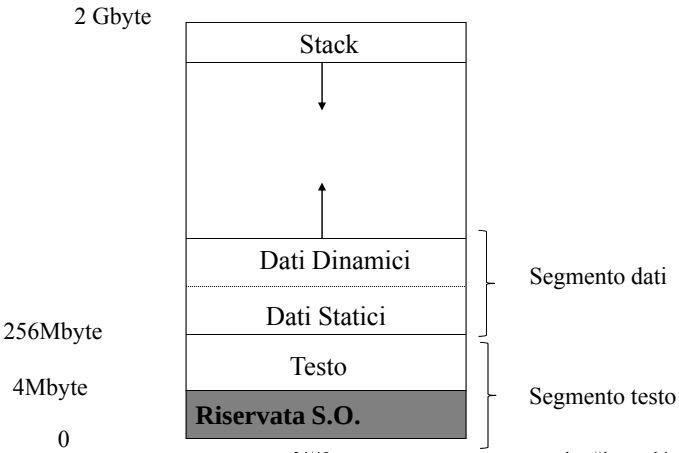
I salti incondizionati

Salti incondizionati assoluti (j, jal...) – Formato J: `j 80000`

Il salto viene sempre eseguito.

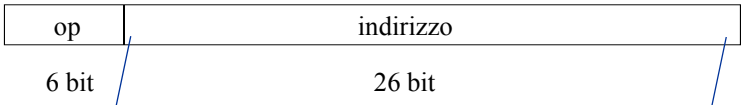
L'indirizzo di destinazione del salto è un indirizzo assoluto di memoria.

L'indirizzo di destinazione del salto è un numero sempre positivo.

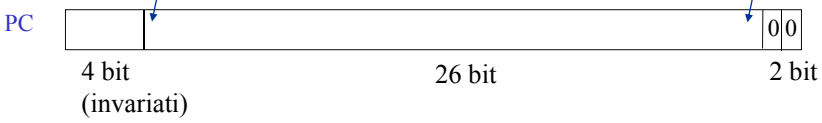


Formato istruzioni di tipo J

- E' il formato usato per le istruzioni di salto incondizionato *(jump)*:



- In questo caso, i campi hanno il seguente significato:
  - **op** indica il tipo di operazione**;**
  - **indirizzo** (composto da **26-bit**) riporta una parte (26 bit su 32) dell'indirizzo **assoluto** di destinazione del salto.
- I 26-bit del campo **indirizzo** rappresentano un indirizzo di parola **(word address)** 2^{26} parole.



Codifica delle istruzioni

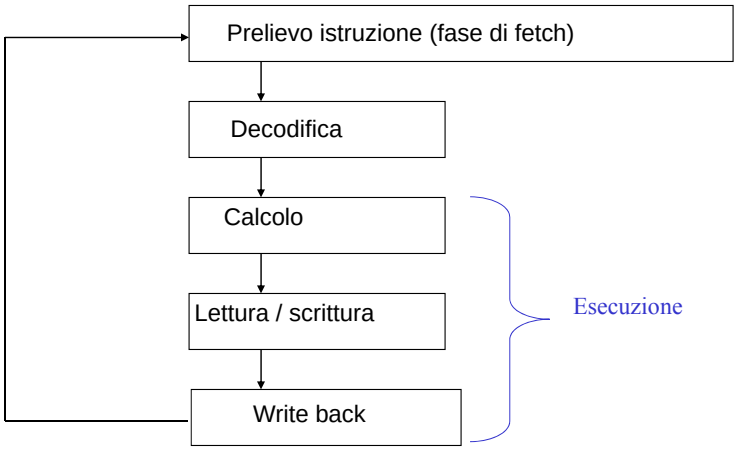
- I 32 bit hanno un significato diverso a seconda del formato (o tipo) di istruzione
  - il tipo di istruzione è riconosciuto in base al valore di alcuni bit **(6 bit)** più significativi **(codice operativo - *OPCODE*)**
- Le istruzioni MIPS sono di **3** tipi (formati):
  - **Tipo R (register) – Lavorano su 3 registri.**
    - Istruzioni aritmetico-logiche.
  - **Tipo I (immediate) – Lavorano su 2 registri. L'istruzione è suddivisa in un gruppo di 16 bit contenenti informazioni + 16 bit riservati ad una costante.**
    - Istruzioni di accesso alla memoria o operazioni contenenti delle costanti.
  - **Tipo J (jump) – Lavora senza registri: codice operativo + indirizzo di salto.**
    - Istruzioni di salto incondizionato.

	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
R	op	rs	rt	rd	shamt	funct
I	op	rs	rt	indirizzo		
J	op	indirizzo				

Introduzione alla CPU

- Introduzione
- CPU

Ciclo di esecuzione di un'istruzione MIPS



I componenti di un'architettura

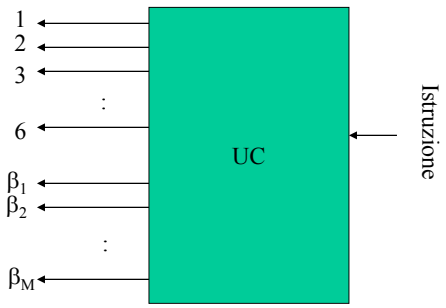
CPU

- Banco di registri *(Register File)* ad accesso rapido, in cui memorizzare i dati di utilizzo più frequente. Il tempo di accesso ai registri è circa 10 volte più veloce del tempo di accesso alla memoria principale.
- Registro *Program counter (PC).* Contiene l'indirizzo dell'istruzione corrente da aggiornare durante l'evoluzione del programma, in modo da prelevare dalla memoria la corretta sequenza di istruzione;
- Registro *Instruction Register (IR)*. Contiene l'istruzione in corso di esecuzione. Questo registro verrà utilizzato più avanti nelle architetture multi-ciclo.
- Unità per l'esecuzione delle operazioni aritmetico-logiche (*Arithmetic Logic Unit - ALU*). I dati forniti all'*ALU* possono provenire da registri oppure direttamente dalla memoria, a seconda delle modalità di indirizzamento previste;
- Unità aggiuntive per elaborazioni particolari come unità aritmetiche per dati in virgola mobile *(Floating Point Unit – FPU)*, sommatori ausiliari, ecc.;
- ***Unità di controllo***. Controlla il flusso e determina le operazioni di ciascun blocco.

MEMORIA PRINCIPALE

L'unità di controllo

- Unità di controllo coordina i flussi di informazione (è il "cervello" della CPU):
- 1) abilitando le vie di comunicazione opportune a seconda dell'istruzione in corso di esecuzione.
- 2) selezionando l'operazione opportuna delle ALU.



Come funziona una CPU?

- Usa un registro, il Program Counter (PC) per ottenere l'indirizzo dell'istruzione.
- Preleva l'istruzione dalla memoria e la inserisce nell'IR.

- Capisce di che tipo di istruzione si tratta (decodifica).
  - usa l'istruzione stessa per decidere cosa fare esattamente.
- Legge il contenuto dei registri.

Da qui le istruzioni si differenziano.

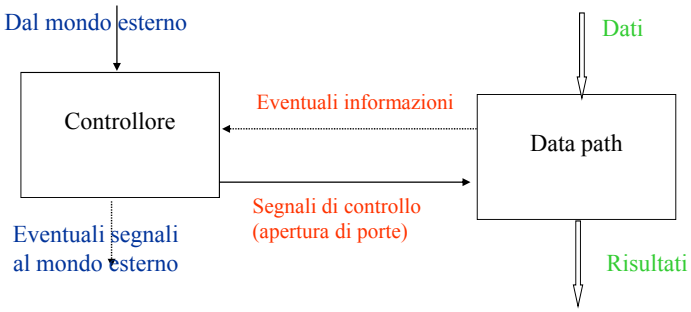
- Calcolo: utilizzo dell'ALU dopo aver letto i registri:
  - per calcolare l'indirizzo in memoria.
  - per eseguire un'operazione logico-aritmetica.
  - per effettuare test (uguaglianza, disuguaglianza, <…).

- Accesso alla memoria.

- Scrittura del risultato nel register file.

Rapporto UC - Dati

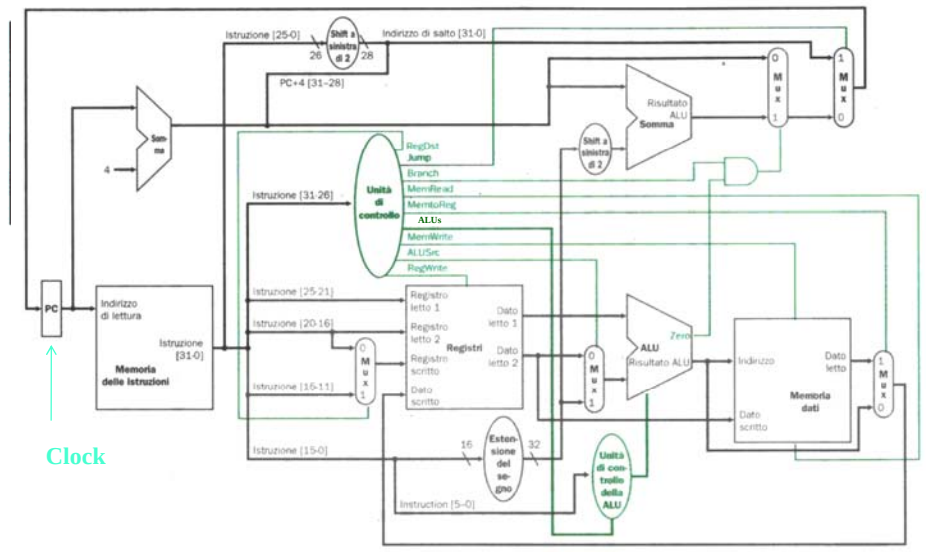
La CPU è un'architettura del tipo: Controllore - Data-path



Fase comune nel ciclo di esecuzione:
•Fase di fetch
•Decodifica (generazione dei segnali di controllo)

Fase diversa: Esecuzione (Calcolo, Accesso memoria, WriteBack)

CPU + UC a singolo ciclo



Istruzioni

`add $s1, $s2, $s3`	000000	10010	10011	10001	00000	100000	
`beq $s1, $s2, -100`	000100	10001	10010	1111	1111	1110	0111
`lw $t0, 32 ($s3)`	100011	10011	01000	0000	0000	0010	0000
`sw $t0, 32 ($s3)`	101011	10011	01000	0000	0000	0010	0000
`addi $t0, $s3, 64`	001000	10011	01000	0000	0000	0100	0000
`j 0x80000`	000010	00 0000 0100 0000 0000 0000 0000					

Introduzione alla CPU

- Introduzione
- CPU