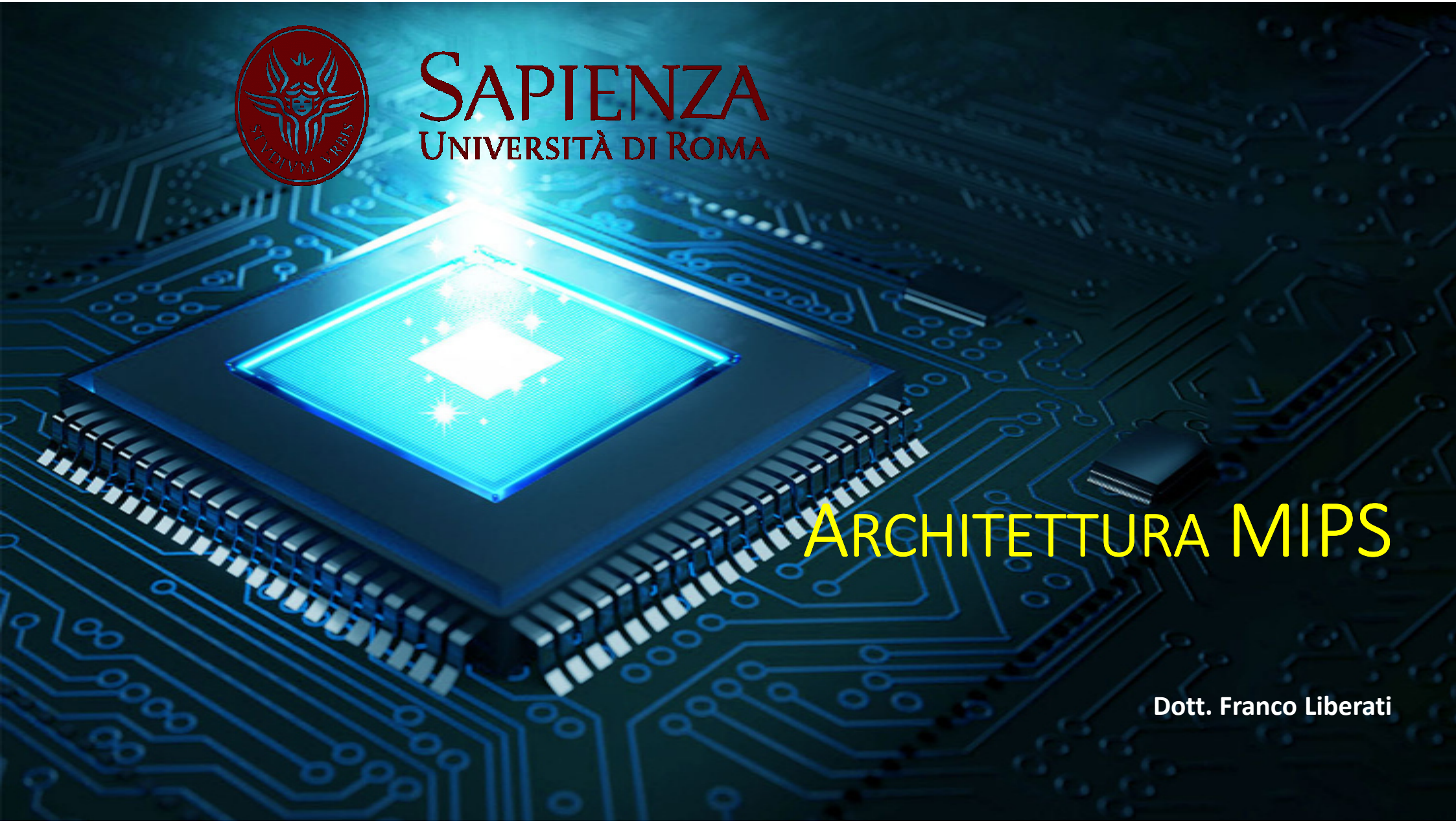
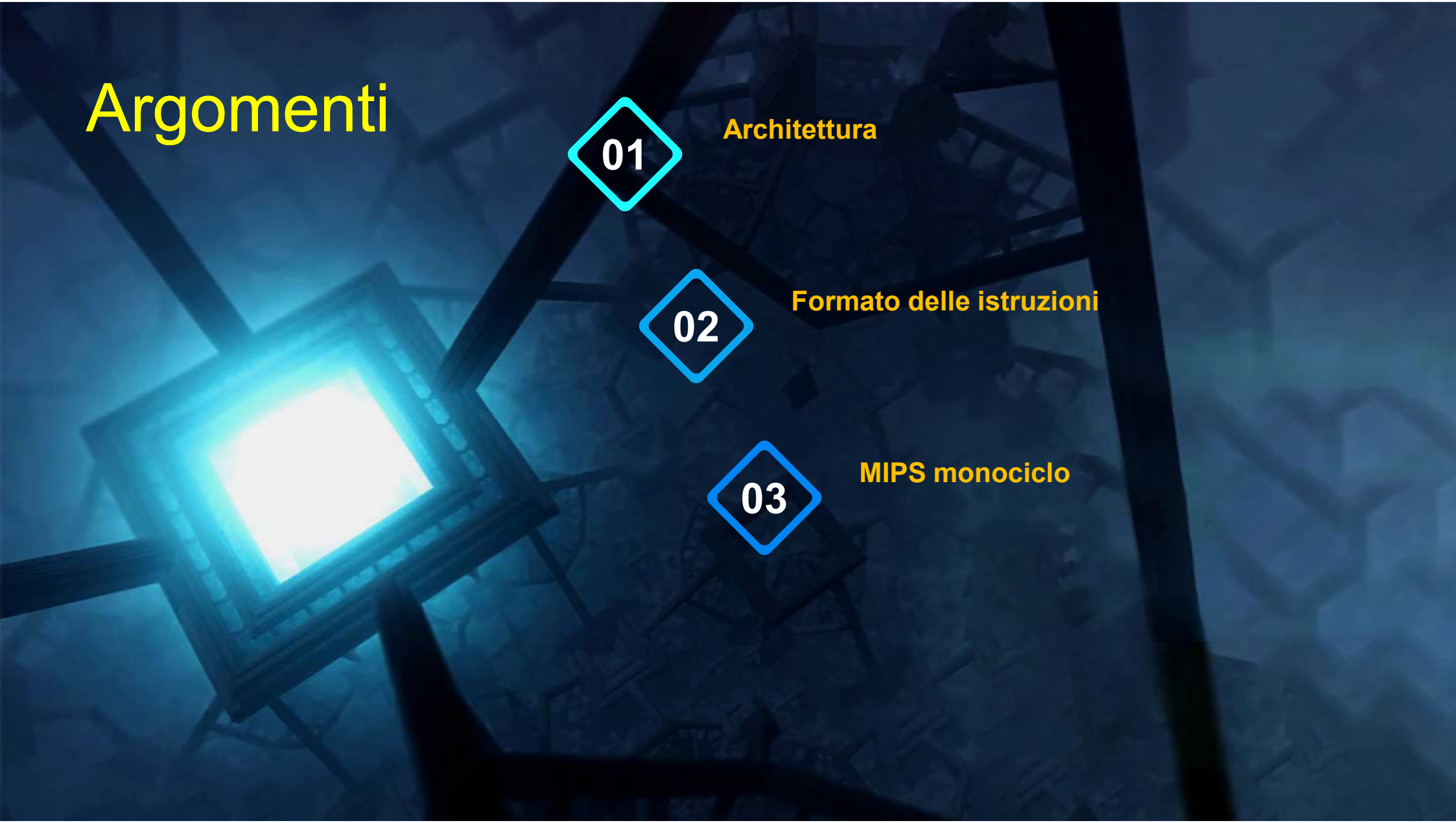
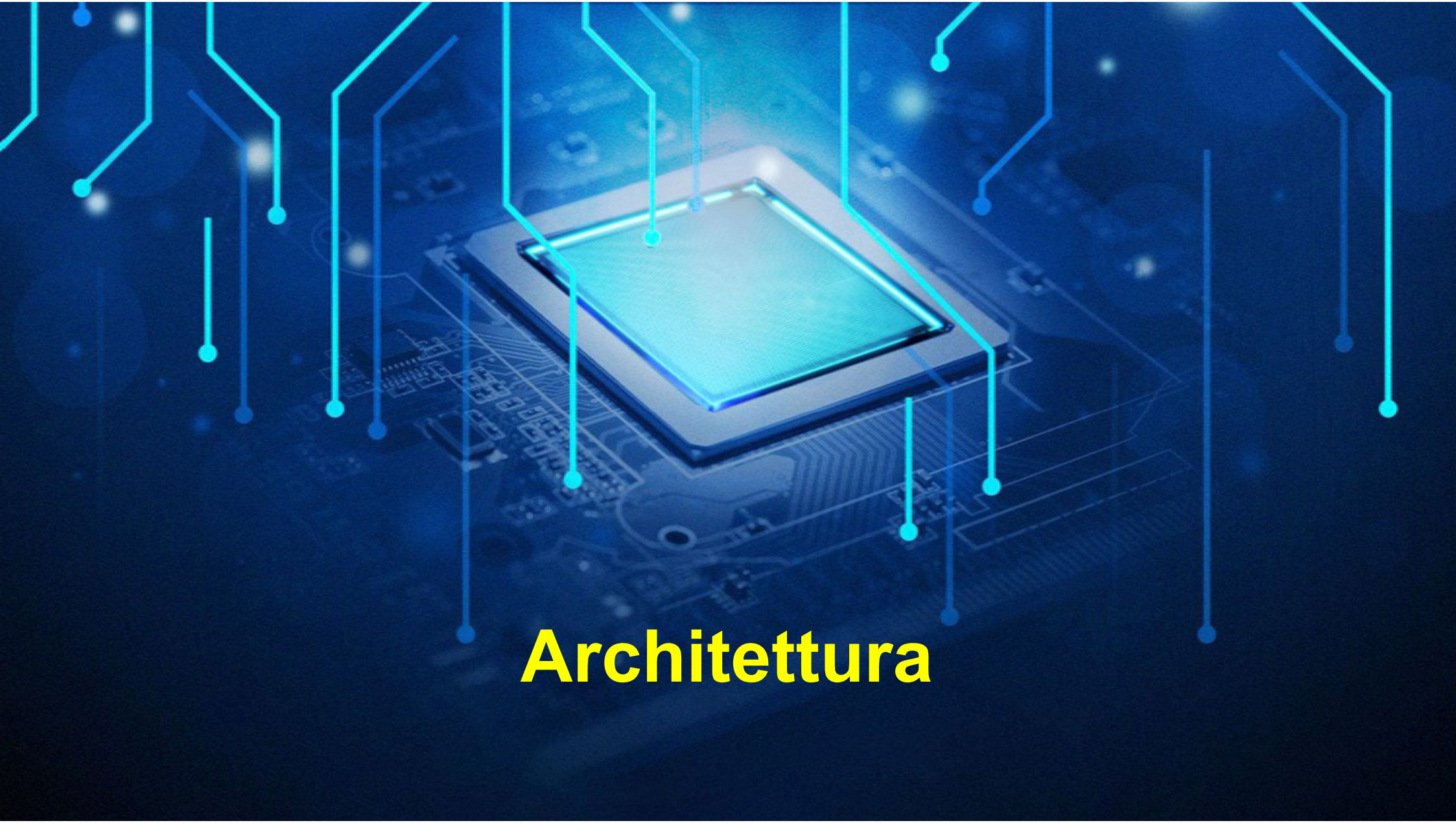


ARCHITETTURA MIPS

Dott. Franco Liberati


Argomenti
01
Architettura
02
Formato delle istruzioni
03
MIPS monociclo


Architettura

Architettura

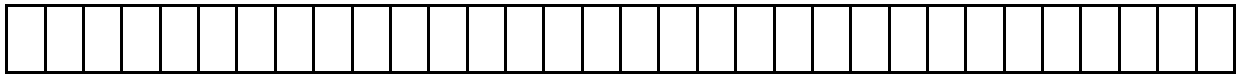
- Il **MIPS** è un processore *general-purpose*, cioè in grado di risolvere algoritmi generici, ed ha una microarchitettura diversa rispetto a quella della Macchina di von Neumann (ha una **suddivisione della memoria in due parti**: una contenente le istruzioni, Instruction Memory, e l'altra i dati, Data Memory), ma preservando i moduli principali (Unità di Controllo, registri ad uso speciale, registri ad uso generale ,Unità di Calcolo, Moduli di I/O)
- Lo studio della programmazione in linguaggio assemblativo MIPS inizia dalla descrizione della sua **microarchitettura**, cioè i componenti interni alla CPU

Architettura

Lunghezza parola

- Il progettista, come primo aspetto, stabilisce quali e quante istruzioni devono essere realizzate (parametri dedotti dalla tecnologia usata, dal compito che deve svolgere la macchina e dai fondi disponibili)
- In seguito il progettista determina **la dimensione della parola** (e quindi il numero di *latch* che devono comporre un registro) e valuta se usare istruzioni a lunghezza fissa, cioè che ogni istruzione risiede in una sola locazione di memoria, o variabile, ovvero che una istruzione può occupare una, due o più locazioni di memoria

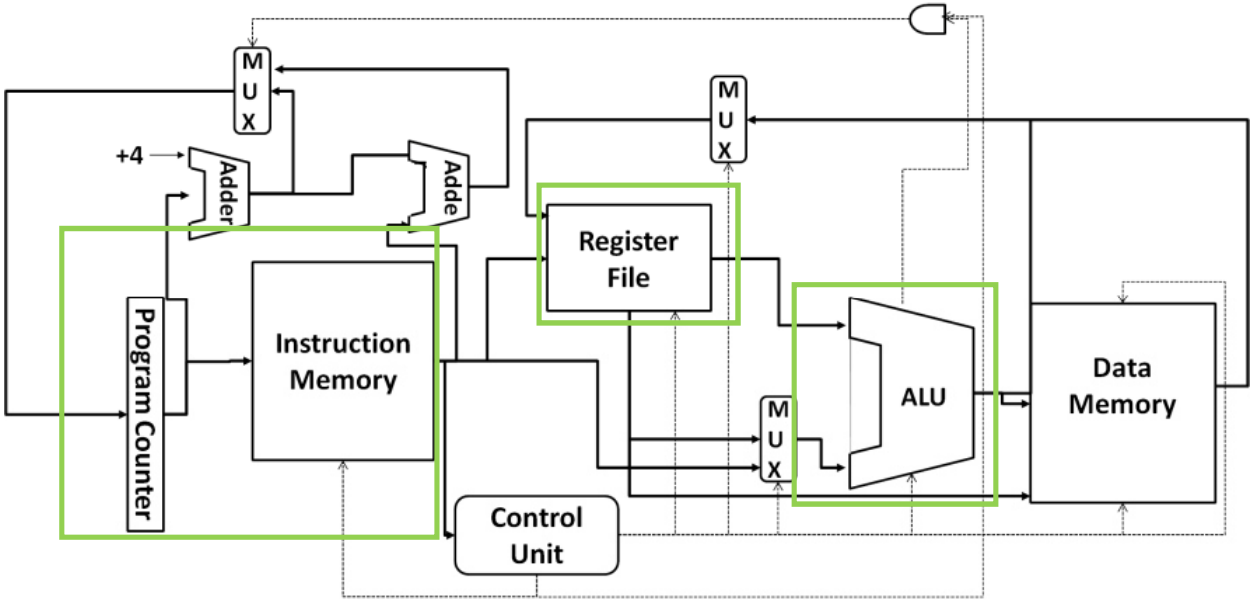
Il MIPS utilizza parole fisse di 32 bit



Architettura

Datapath

- Come passo successivo si stabilisce il **formato delle istruzioni** e si progettano le **componenti elettroniche** (cioè le reti sequenziali e quelle combinatorie) che formano l'Unità di Calcolo e l'Unità di Controllo
  - L'**insieme delle istruzioni macchina** di un processore è detto *Instruction Set Architecture* (ISA)
- Questi due elementi determinano il **percorso dei dati** (*datapath*), ovvero il passaggio che deve compiere una istruzione per essere elaborata
- Il percorso dei dati è attraversato completamente, o in parte, in relazione al formato dell'istruzione. In generale una istruzione richiede un maggior tempo di espletamento se deve attraversare l'intero percorso dei dati (perché passa per più circuiti elettronici che inducono un ritardo implicito)

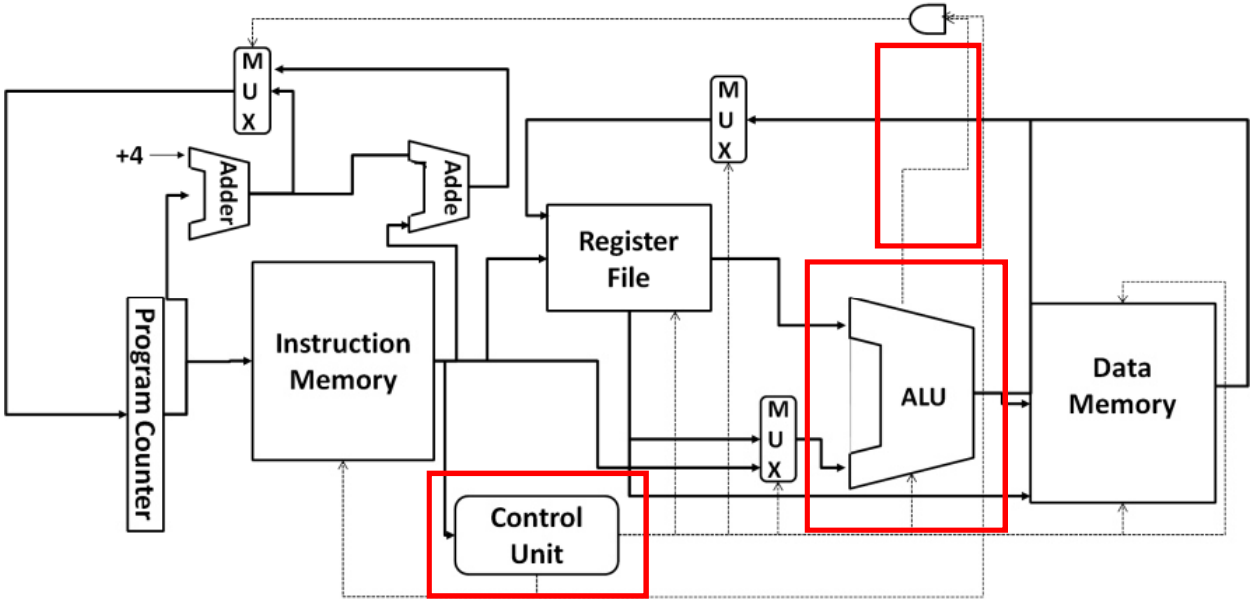


ADD $t3,$t2,$t1

Architettura

Descrizione MIPS: UC- ALU

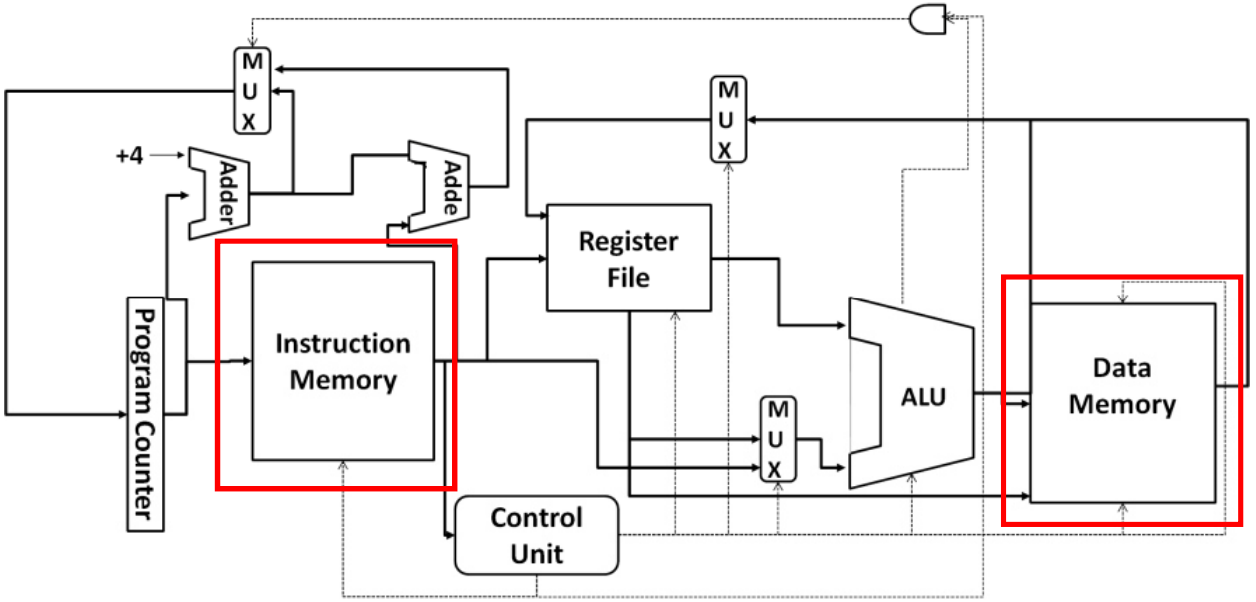
- L'**Unità di Controllo** (*Control Unit*) genera comandi utili per prelevare ed elaborare le istruzioni. È realizzata con una rete combinatoria (nel caso del MIPS monociclo) o sequenziale (se si prevede la canalizzazione, MIPS multiciclo)
- L'**Unità Logica Aritmetica** (ALU) è il modulo deputato a svolgere le principali funzioni matematiche, tra numeri interi binari, e logiche, per stringhe binarie.
- Alcune linee di uscita dalla ALU riportano i **codici di condizione** (*condition code*) che sono utili per svolgere i salti condizionali (aspetto fondamentale delle macchine programmabili; a differenza delle programmate).



Architettura

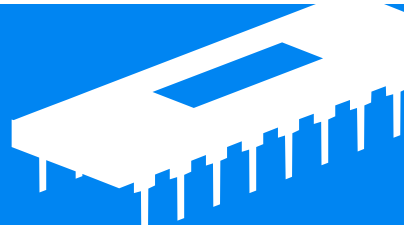
Descrizione MIPS: Memoria Dati e Memoria Istruzioni

- La Memoria non ha una struttura monolitica ma è fisicamente suddivisa in due parti disgiunte e distinte: la **Memoria delle Istruzioni** (*Instruction Memory*), in cui è archiviato il programma, e la **Memoria dei Dati** (*Data Memory*), dove risiedono gli operandi da elaborare o i risultati ottenuti in seguito alle operazioni logiche e aritmetiche
- Infine ci sono i **Dispositivi di Ingresso e di Uscita** (periferiche) che sono collegati con l'Unità di Elaborazione e con le due Unità di Memoria attraverso una struttura di interconnessione basata su pluri-bus. Tra le periferiche c'è il **coprocessore matematico**, una circuiteria in grado di svolgere operazioni con numeri reali (*floating point*)



Architettura

Registri della CPU



- All'interno dell'Unità di Elaborazione del MIPS sono presenti trentadue registri, di cui ventiquattro sono **ad uso generale** (*Register File*) e sono identificati dai simboli $v*n*, $s*n* e $t*n*, mentre gli altri sono **ad uso speciale**

- I registri sono identificati dal simbolo $ seguito da un numero. Per venire incontro al programmatore si può usare, durante la fase di programmazione in linguaggio assemblativo, un acronimo, di solito due lettere o una lettera ed un numero (es.:$t0, $v0, $ra,…)

Registri del MIPS

Numero del registro	Nome convenzionale del registro	Uso
$0	$zero	Registro non modificabile con valore impostato a zero
$1	$at	Riservato per risolvere le pseudoistruzioni
$2, $3	$v0, $v1	Valori di ritorno da subroutine
$4-$7	$a0-$a3	Parametri di ingresso (argomenti) per la subroutine
$8-$15	$t0-$t7	Registri temporanei non preservanti (il contenuto attivando una chiamata a subroutine è impostato a 0)
$16 - $23	$s0-$s7	Registri temporanei preservanti (conservano il contenuto dopo una chiamata a subroutine)
$24, $25	$t8, $t9	Registri temporanei non preservanti (il contenuto dopo una chiamata a subroutine è impostato a 0)
$26, $27	$k0, $k1	Riservati per il kernel
$28	$gp	Puntatore alla zona di memoria con dati condivisi (Global Area Pointer)
$29	$sp	Puntatore allo stack (Stack Pointer)
$30	$fp	Puntatore al frame (Frame Pointer)
$31	$ra	Registro per il ritorno da subroutine (Return Address)

Architettura

Registri della CPU

- I **registri ad uso speciale** hanno analogie con quelli presenti nella macchina di von Neumann (e con le successive modifiche approntate ad essa per potenziarne l'efficienza e l'efficacia)
- I **registri ad uso generale** svolgono un ruolo utile perché consentono di memorizzare operandi e indirizzi, evitando in questo modo continui e lenti accessi alla Memoria dei Dati, ed offrono l'opportunità di lavorare con istruzioni a lunghezza fissa, minimizzando così gli accessi alla Memoria delle Istruzioni

Registri del MIPS

Numero del registro	Nome convenzionale del registro	Uso
$0	$zero	Registro non modificabile con valore impostato a zero
$1	$at	Riservato per risolvere le pseudoistruzioni
$2, $3	$v0, $v1	Valori di ritorno da subroutine
$4-$7	$a0-$a3	Parametri di ingresso (argomenti) per la subroutine
$8-$15	$t0-$t7	Registri temporanei non preservanti (il contenuto attivando una chiamata a subroutine è impostato a 0)
$16 - $23	$s0-$s7	Registri temporanei preservanti (conservano il contenuto dopo una chiamata a subroutine)
$24, $25	$t8, $t9	Registri temporanei non preservanti (il contenuto dopo una chiamata a subroutine è impostato a 0)
$26, $27	$k0, $k1	Riservati per il kernel
$28	$gp	Puntatore alla zona di memoria con dati condivisi (Global Area Pointer)
$29	$sp	Puntatore allo stack (Stack Pointer)
$30	$fp	Puntatore al frame (Frame Pointer)
$31	$ra	Registro per il ritorno da subroutine (Return Address)

Architettura

Registri della CPU

- Tra i **registri ad uso speciale** devono essere citati
  - Il **Contatore di Programma** (*Program Counter*, pc)
  - il **Puntatore a Stack** (*Stack Pointer,* $sp)
  - Un **registro per il ritorno da subroutine** (*Return Address*, $ra)
  - Il **Registro di Stato** (*Status Register*, $12 del CP0, status)
  - Cinque **Registri per la gestione delle Interruzioni** ($k0; $k1; $13 del Cp0, Cause; $14 del CP0, Epc; $8 del CP0, BadVAddr)
  - il Puntatore alla porzione di area di Memoria dei Dati, in cui risiedono delle informazioni comuni (*Global Area Pointer*, $gp)
  - il Puntatore al Frame (*Frame Pointer*, $fp)

Tab.1.1 - Registri del MIPS

Numero del registro	Nome convenzionale del registro	Uso
$0	$zero	Registro non modificabile con valore impostato a zero
$1	$at	Riservato per risolvere le pseudoistruzioni
$2, $3	$v0, $v1	Valori di ritorno da subroutine
$4-$7	$a0-$a3	Parametri di ingresso (argomenti) per la subroutine
$8-$15	$t0-$t7	Registri temporanei non preservanti (il contenuto attivando una chiamata a subroutine è impostato a 0)
$16 - $23	$s0-$s7	Registri temporanei preservanti (conservano il contenuto dopo una chiamata a subroutine)
$24, $25	$t8, $t9	Registri temporanei non preservanti (il contenuto dopo una chiamata a subroutine è impostato a 0)
$26, $27	$k0, $k1	Riservati per il kernel
$28	$gp	Puntatore alla zona di memoria con dati condivisi (Global Area Pointer)
$29	$sp	Puntatore allo stack (Stack Pointer)
$30	$fp	Puntatore al frame (Frame Pointer)
$31	$ra	Registro per il ritorno da subroutine (Return Address)

Formato delle Istruzioni

Formato delle Istruzioni

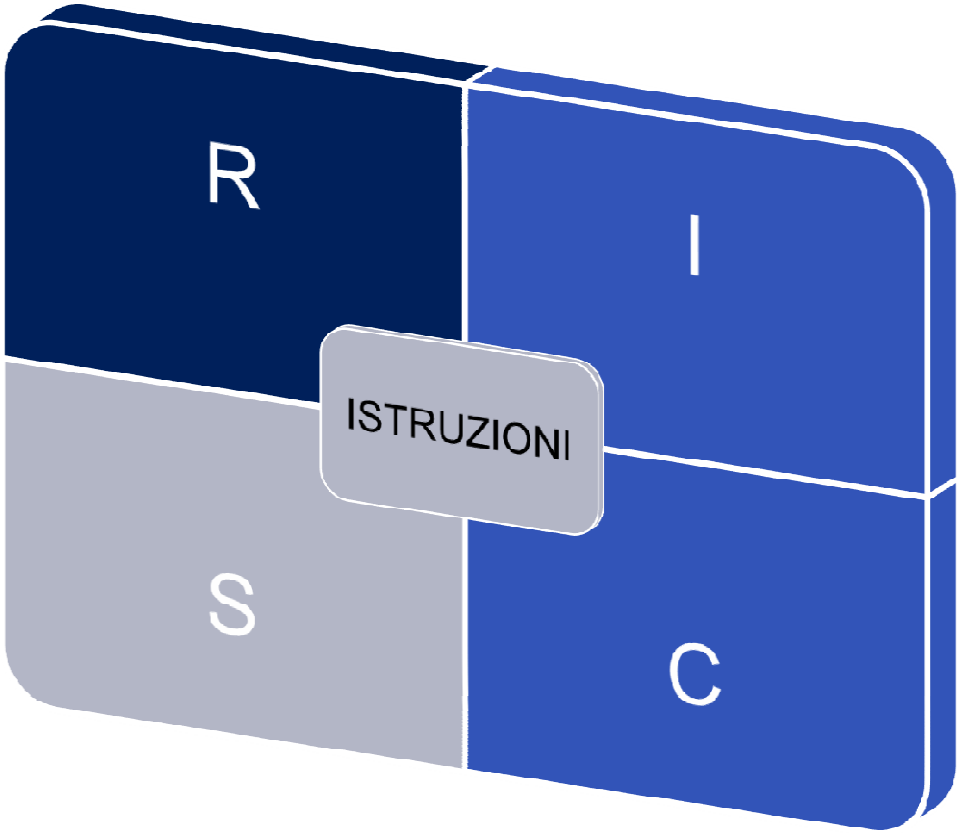
Generalità

- Una istruzione è un codice rappresentato da una stringa binaria suddivisa in due campi principali: il **Codice Operativo** (Opcode), che specifica il formato (ovvero come sono organizzati i bit dell'istruzione e quale è il significato logico della stessa), e il **Modo di Indirizzamento** (Addressing Mode), che indica un indirizzo interno al programma oppure il luogo (l'istruzione stessa, un registro o una locazione di memoria) in cui risiedono gli operandi (registri, locazione, istruzione stessa – *immediate*).
- Il processore MIPS ha istruzioni a lunghezza fissa a 32bit (di cui l'Opcode occupa i primi sei bit più significativi). La lunghezza fissa comporta il **vantaggio** di avere una semplice ed immediata fase di prelievo; un datapath di scarsa estensione; e, di conseguenza, un minor utilizzo di componenti, una maggiore velocità di esecuzione (rispetto ad un architettura CISC), un ridotto consumo di energia e un economico costo di fabbricazione

Formato delle Istruzioni

Formato delle Istruzioni MIPS

- Il processore MIPS gestisce **quattro formati di istruzioni**: a registro (R), a valore immediato (I), di salto (S) e inerenti il coprocessore matematico (C)
- A loro volta le istruzioni hanno una **suddivisione in classi** in accordo ai compiti svolti: spostamento; operazione logica-aritmetica; salto condizionato o incondizionato; e, infine, di sistema.



Formato delle Istruzioni

Tipo R

Le **istruzioni di tipo R** sono caratterizzate dal modo di indirizzamento a registro con una organizzazione che prevede due operandi siti in altrettanti registri, denominati sorgenti (SOURCE1 e SOURCE2), a cui si applica una funzione logico-aritmetica (FUNCTION) e il cui risultato, proveniente dalla ALU, è copiato in un registro di destinazione (DESTINATION). Nello specifico i sei bit più significativi del formato R, il Codice Operativo (OPCODE), sono impostati a zero e indicano all'Unità di Controllo che si tratta di una istruzione di tipo R; mentre nei primi sei bit meno significativi (FUNCTION) si specifica l'operazione logico-aritmetica che deve essere applicata agli operandi siti nei registri sorgente (bit 21-25 e bit 16-20)

Formato delle istruzioni di tipo R nel MIPS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OPCODE						SOURCE1					SOURCE2					DESTINATION					*unused*					FUNCTION					
0	0	0	0	0	0	b	b	b	b	b	b	b	b	b	b	b	b	b	b	b	0	0	0	0	0	f	f	f	f	f	f

Formato delle istruzioni di tipo R nel MIPS (shift e rotate)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OPCODE						SOURCE1					SOURCE2					DESTINATION					SHIFT					FUNCTION					
0	0	0	0	0	0	b	b	b	b	b	b	b	b	b	b	b	b	b	b	b	s	s	s	s	s	f	f	f	f	f	f

Formato delle Istruzioni

Tipo R: esempio

Una istruzione che sfrutta questo formato è l'addizione, **add** (in generale sono le istruzioni logiche-aritmetiche), che richiede due addendi, presenti nei registri sorgenti, e riporta la somma nel registro di destinazione

add $t3,$t2,$t1 **#$t3←$t2+$t1**

Istruzione di tipo R nel MIPS e sua traduzione in linguaggio macchina					
Istruzione in linguaggio asemblativo canonico					
add	$t3,	$t2,	$t1		
Istruzione in linguaggio assemblativo nativo					
R	$10,	$9,	$11,	00000	ADD
Istruzione in linguaggio macchina					
000000	01010	01001	01011	00000	100000

Formato delle Istruzioni

Tipo I

Le **istruzioni di tipo I** includono al loro interno un operando; si realizza cioè un modo di indirizzamento immediato (*immediate*). In questo formato, infatti, è possibile specificare un valore senza fare riferimento (e quindi accesso) ad una locazione della Memoria dei Dati.
I sei bit più significativi (OPCODE) indicano all'Unità di Controllo che si tratta di una istruzione di tipo I; mentre nei primi sedici bit (IMMEDIATE) risiede l'operando o l'indirizzo in cui risiede l'operando

Formato delle istruzioni di tipo I nel MIPS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OPCODE						SOURCE					DESTINATION					IMMEDIATE																
x	x	x	x	x	x	b	b	b	b	b	b	b	b	b	b	i	i	i	i	i	i	i	i	i	i	i	i	i	i	i	i

I 16bit riservati al campo IMMEDIATE consentono di rappresentare un intervallo di valori interi piuttosto scarso [-32768;+32767]. Questo limite, per ora, non pregiudica la progettazione dell'Unità di Elaborazione che si intende descrivere (e quindi se ne trascurano gli effetti collaterali).

Formato delle Istruzioni

Tipo I: esempio

Tale formato consente di svolgere operazioni logiche-aritmetiche senza accessi in memoria. Ad esempio si sfrutta l'addizione con un valore immediato, addi (Esempio 1.2), che prevede l'uso dei registri ausiliari SOURCE, per memorizzare il secondo operando, e DESTINATION, in cui riportare la somma risultante

addi $t3,$t4,7 #$t3←$t4+7

Istruzione di tipo I nel MIPS (somma diretta) e sua traduzione in linguaggio macchina			
Istruzione in linguaggio asemblativo canonico			
addi	$t3,	$t4,	7
Istruzione in linguaggio assemblativo nativo			
I	$12,	$11,	7
Istruzione in linguaggio macchina			
001000	01100	01011	0000000000000111

Formato delle Istruzioni

Tipo I: istruzione di inizializzazione di un registro

Questo formato, inoltre, realizza l'istruzione di caricamento di un operando in un registro (*load immediate*, li); cioè una **inizializzazione**. In altre parole si inserisce un valore in un registro senza esplicitare un indirizzo di memoria (o evitando una modalità più complessa).
L'istruzione di **caricamento di un valore immediato**, che ha una sintassi del tipo **li $reg,immediate**, ovvero la cancellazione del contenuto del registro *$reg* con il numero *immediate*, è riscritta dall'assemblatore sfruttando l'istruzione di addizione senza l'estensione del segno, ***addiu***

li $t3,7 #$t3←7

Istruzione di tipo I nel MIPS (inizializzazione) e sua traduzione in linguaggio macchina			
Istruzione in linguaggio asemblativo canonico			
li	$t3,		7
Istruzione in linguaggio assemblativo nativo			
addiu	$0,	$11	7
Istruzione in linguaggio macchina			
001001	00000	01011	0000000000000111

Formato delle Istruzioni

Tipo I: trasferimento dalla memoria ai registri

Inoltre il formato I, grazie ai campi SOURCE e DESTINATION, concretizza le **istruzioni di trasferimento di operandi dalla (*Load*) e verso (*Store*) la Memoria dei Dati** sfruttando il modo di indirizzamento a registro indiretto con spiazzamento.
L'istruzione lw $t3,8($t4) copia nel registro $t3, il contenuto di una locazione di memoria di quattro byte il cui indirizzo nella Memoria dei Dati è determinato dalla somma del valore presente nel registro $t4 incrementato di 8. Pertanto se nel registro $t4 è immagazzinato il numero 268501000, a questo si somma 8 e si ottiene l'indirizzo, 268501008, della locazione della Memoria dei Dati dove risiede l'operando

lw $t3,8($t4) #$t3←MEM($t4+8)

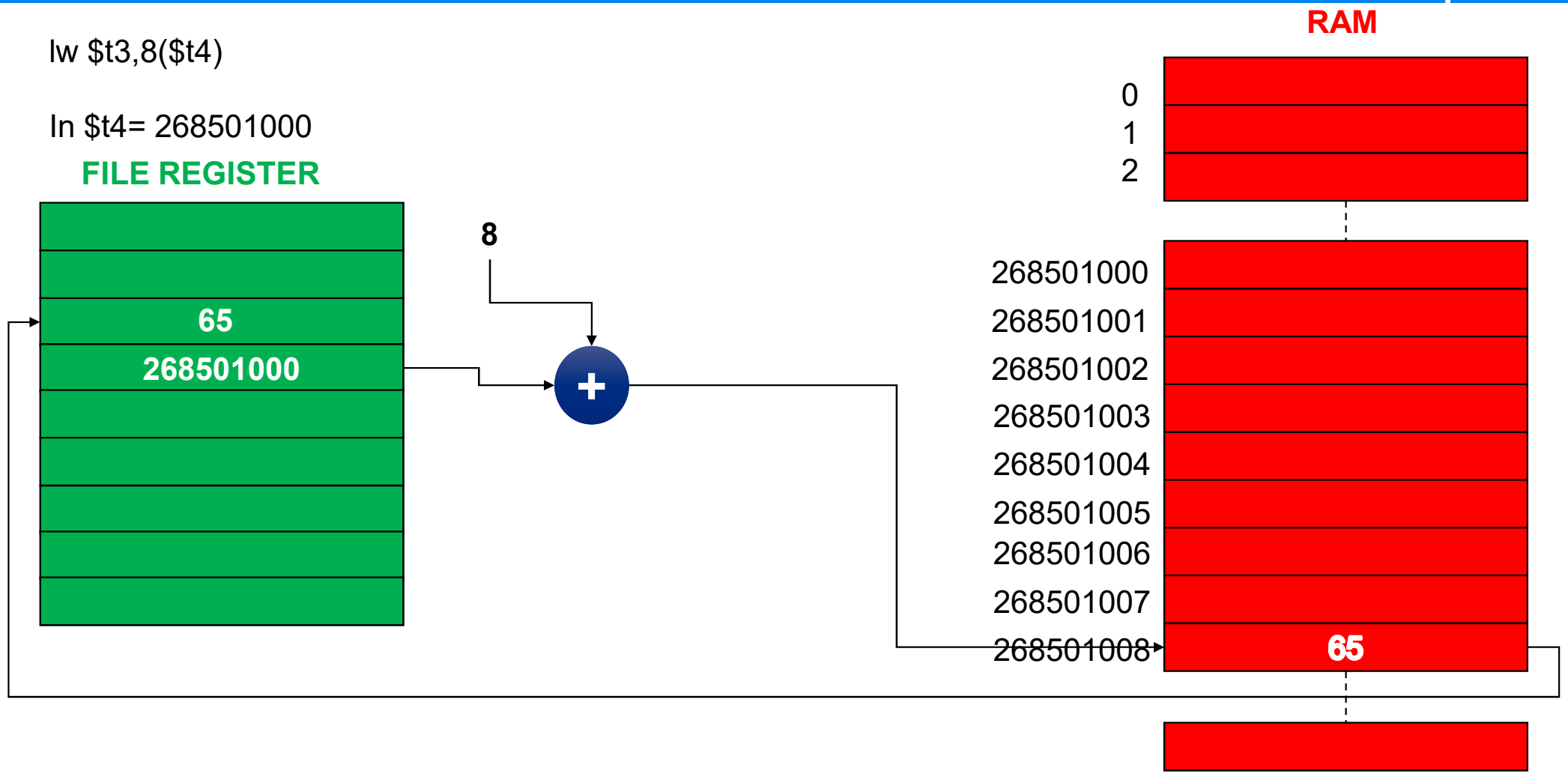
Istruzione di tipo I nel MIPS (trasferimento) e sua traduzione in linguaggio macchina			
Istruzione in linguaggio assemblativo canonico			
lw	$t3,	8($t4)	
Istruzione in linguaggio assemblativo nativo			
lw	$12,	$11	+8
Istruzione in linguaggio macchina			
100011	01100	01011	0000000000001000

Formato delle Istruzioni

Tipo I: trasferimento dalla memoria ai registri

lw $t3,8($t4)

In $t4= 268501000



Formato delle Istruzioni

Tipo I: trasferimento dal registro alla memoria

Analogamente è possibile realizzare l'archiviazione di un operando, contenuto in un registro ,nella Memoria dei Dati.
L'istruzione sw $t3,8($t4) copia il contenuto del registro $t3 in quattro byte della Memoria dei Dati che si trovano a partire dall'indirizzo ricavato dalla somma del valore 8 con il numero presente nel registro $t4.

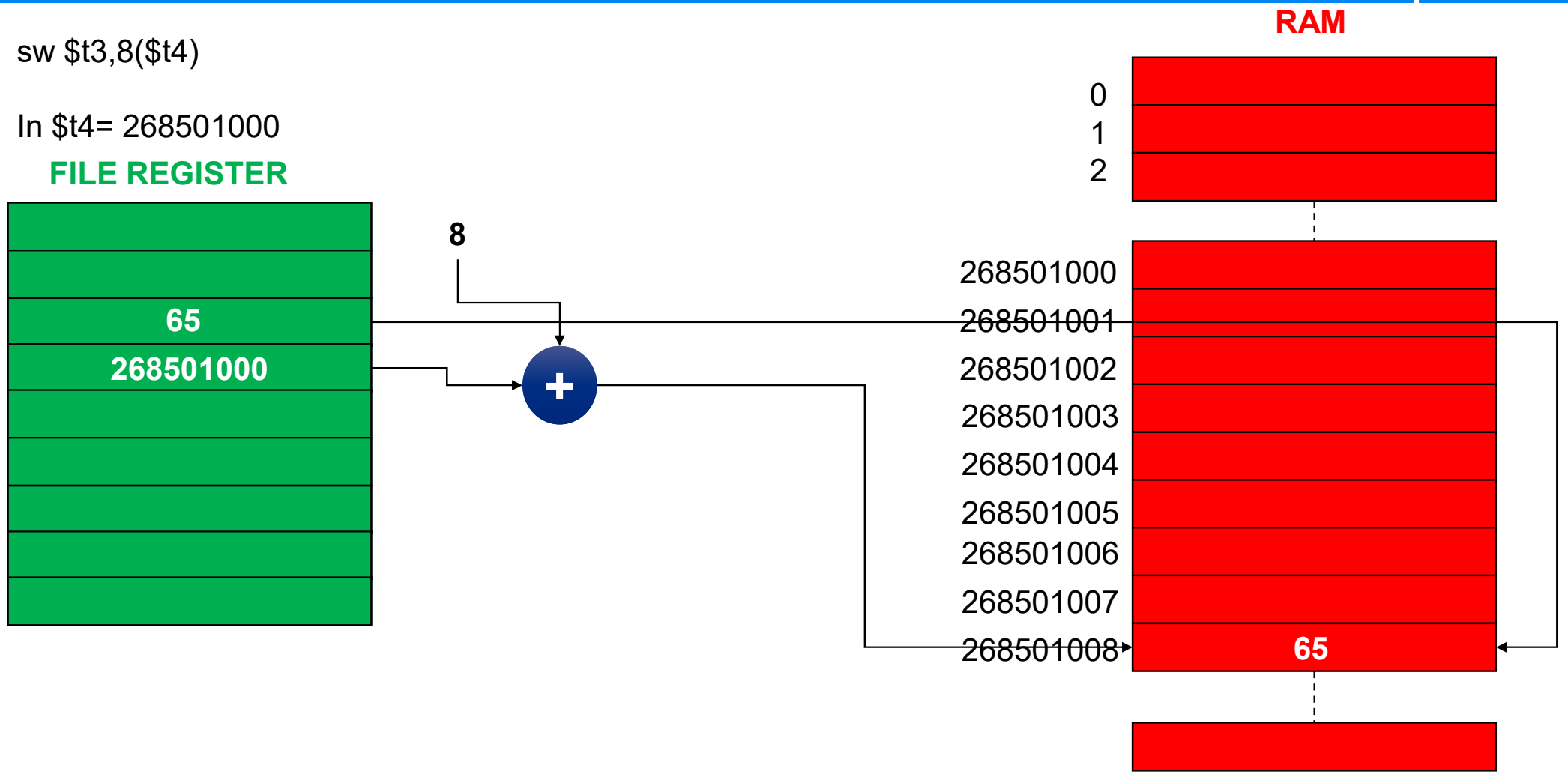
Istruzione di tipo I nel MIPS (trasferimento) e sua traduzione in linguaggio macchina			
Istruzione in linguaggio assemblativo canonico			
sw	$t3,	8($t4)	
Istruzione in linguaggio assemblativo nativo			
l	$12,	$11	+8
Istruzione in linguaggio macchina			
101011	01100	01011	0000000000001000

Formato delle Istruzioni

Tipo I: trasferimento dalla memoria ai registri

sw $t3,8($t4)

In $t4= 268501000



Formato delle Istruzioni

Tipo I: trasferimento dalla memoria ai registri (OSSERVAZIONE)

La comun istruzione lw <registrodestinazione>,<etichetta> è così: gestita

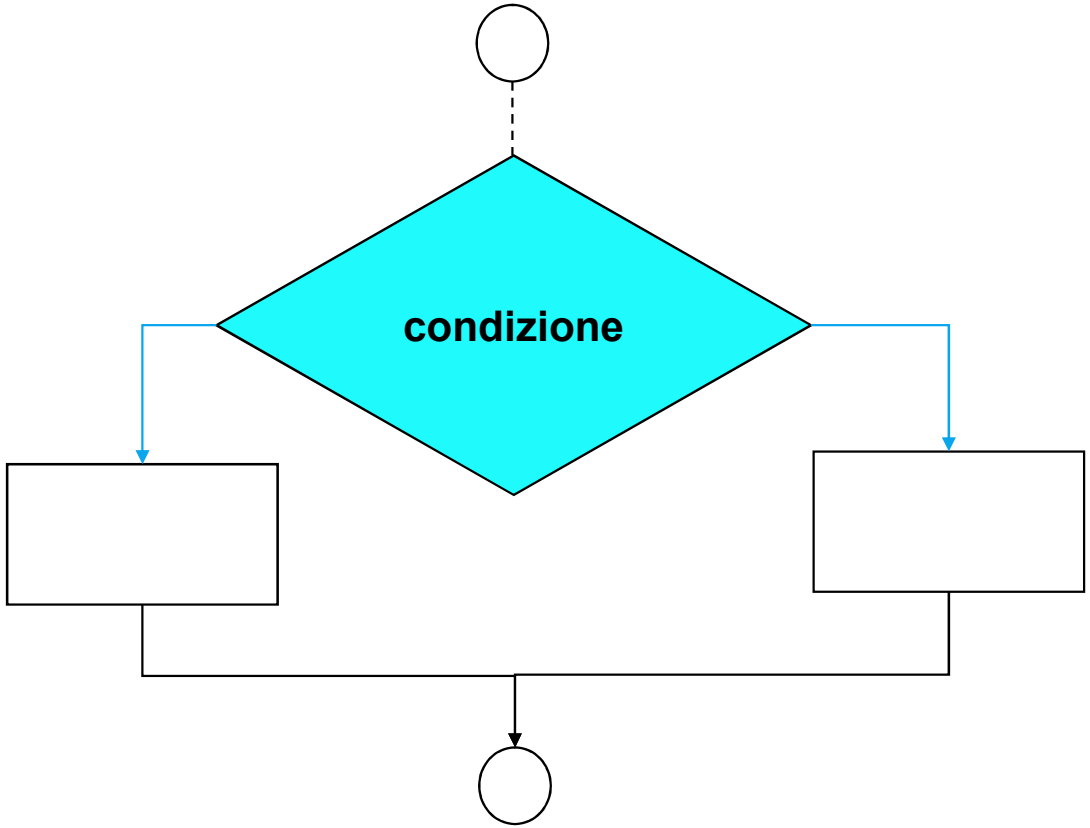
lw $t0,pippo #$t3←MEM(locazione_pippo)

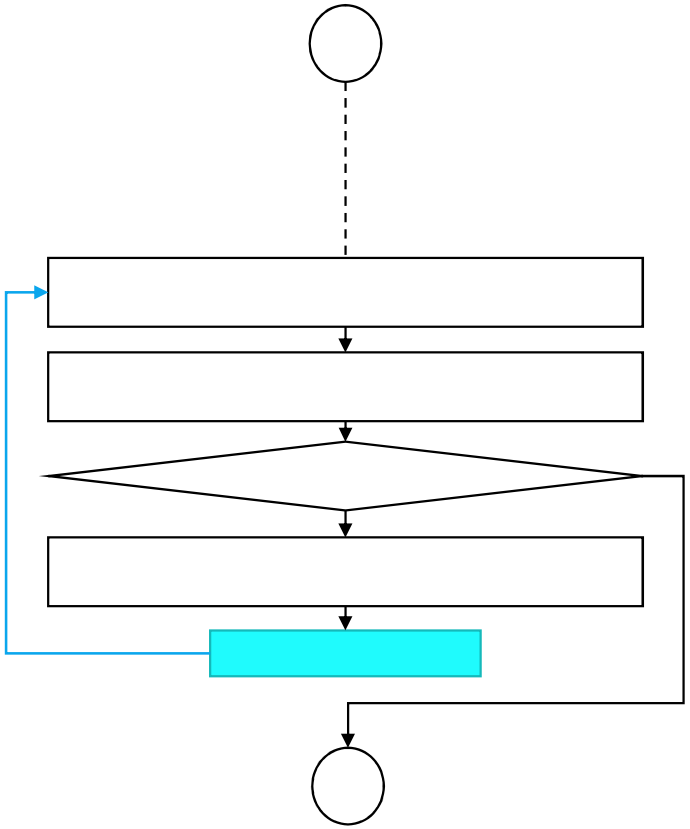
Istruzione di tipo I nel MIPS (trasferimento) e sua traduzione in linguaggio macchina			
Istruzione in linguaggio assemblativo canonico			
lw	$t0,	pippo	#pippo è la locazione 0000100000000001 0000000000000000
Istruzione in linguaggio assemblativo nativo			
lui	$at,		4097
lw	$at	$8	0
Istruzione in linguaggio macchina			
100011	01100	01011	0000000000001000

Formato delle Istruzioni

Tipo S

Le **istruzioni di tipo S** sono quelle relative ai salti e si differenziano in due sotto formati: **salto condizionato** e **incondizionato**.





Formato delle Istruzioni

Tipo S: salto condizionato

Il **salto condizionato** (*branch*) ha un formato in cui si specificano due registri (REG1 e REG2) il cui contenuto è analizzato e nel caso di veridicità di una condizione, si procede all'istruzione sita all'indirizzo specificato nel campo OFFSET (per poi proseguire sequenzialmente).

Formato delle istruzioni di tipo S (salto condizionato) nel MIPS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OPCODE						REG 1					REG 2					OFFSET															
x	x	x	x	x	x	b	b	b	b	b	b	b	b	b	b	i	i	i	i	i	i	i	i	i	i	i	i	i	i	i	i

beq **000100** #Salto se i due registri hanno operando uguali
bgez **000001** #Salto se il registro ha un operando maggiore o uguale a zero
blez **000110** #Salto se il registro ha un operando minore o uguale a zero

Formato delle Istruzioni

Tipo S: salto condizionato

Questo formato di istruzione sfrutta delle linee che fuoriescono dalla Unità Logico-Artimetica, note come **codici di condizione** (o *flag*). Un flag è una linea di controllo che può essere attiva (asserita), oppure no, in accordo all'ultima operazione svolta.
L'asserzione di un flag (o di una combinazione di essi) comporta la copia dell'indirizzo di salto all'interno del Contatore di Programma, interrompendo in questo modo la sequenzialità delle istruzioni.
Un esempio è quello di un salto condizionato alla locazione 60000 della Memoria delle Istruzioni quando gli operandi siti in due registri sono uguali. Con $t3=11 e $t4=11 il confronto degli operandi attiva il salto perché alla fine della sottrazione dei due operandi, per valutarne l'uguaglianza, il flag Z è asserito.

Istruzione salto condizionato e sua traduzione in linguaggio macchina ($t3=11 e $t4=11)			
Istruzione in linguaggio asemblativo canonico			
beq	$t3,	$t4,	60000
Istruzione in linguaggio assemblativo nativo			
S	$11,	$12	60000
Istruzione in linguaggio macchina			
000100	01011	01100	0011101010011000

Formato delle Istruzioni

Tipo S: salto incodizionato

Il **salto incondizionato** (*jump*) ha un formato più semplice poiché è costituito da un OPCODE e un indirizzo specificato nel campo OFFSET.

Formato delle istruzioni di tipo S (salto incondizionato) nel MIPS																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OPCODE						OFFSET																									
0	0	0	0	1	0	i	i	i	i	i	i	i	i	i	i	i	i	i	i	i	i	i	l	i	i	i	i	i	i	i	i

Formato delle Istruzioni

Un salto incondizionato ad un indirizzo di Memoria delle Istruzioni non richiede alcun confronto tra operandi e non impiega l'Unità Logico-Aritmetica né

Istruzione salto incondizionato e sua traduzione in linguaggio macchina	
Istruzione in linguaggio asemblativo canonico	
j	75000
Istruzione in linguaggio assemblativo nativo	
S	75000
Istruzione in linguaggio macchina	
000010	00000000010010010011111000

L'indirizzo che specifica la locazione a cui saltare è di 26bit, inferiore alla lunghezza della parola che è di 32bit, limitando la capacità di salto (un limite accettabile visto che sono pochi i programmi che hanno tale dimensione e, qualora se ne abbia bisogno, si possono svolgere più salti).
Per poter saltare realmente di 2^{26} istruzioni l'indirizzo è esteso a livello fisico con due bit posti nelle cifre meno significative.
Questa modifica è necessaria perché le memorie sono organizzate fisicamente in locazioni di 8bit e nella Memoria delle Istruzioni, ciascuna istruzione è lunga 32bit e quindi risiede in indirizzi multipli di quattro. Pertanto un salto incondizionato permette di raggiungere 2^{28}(268435456) indirizzi nella Memoria dei Dati da 8bit o 2^{26} (67108864) indirizzi nella Memoria delle Istruzioni.

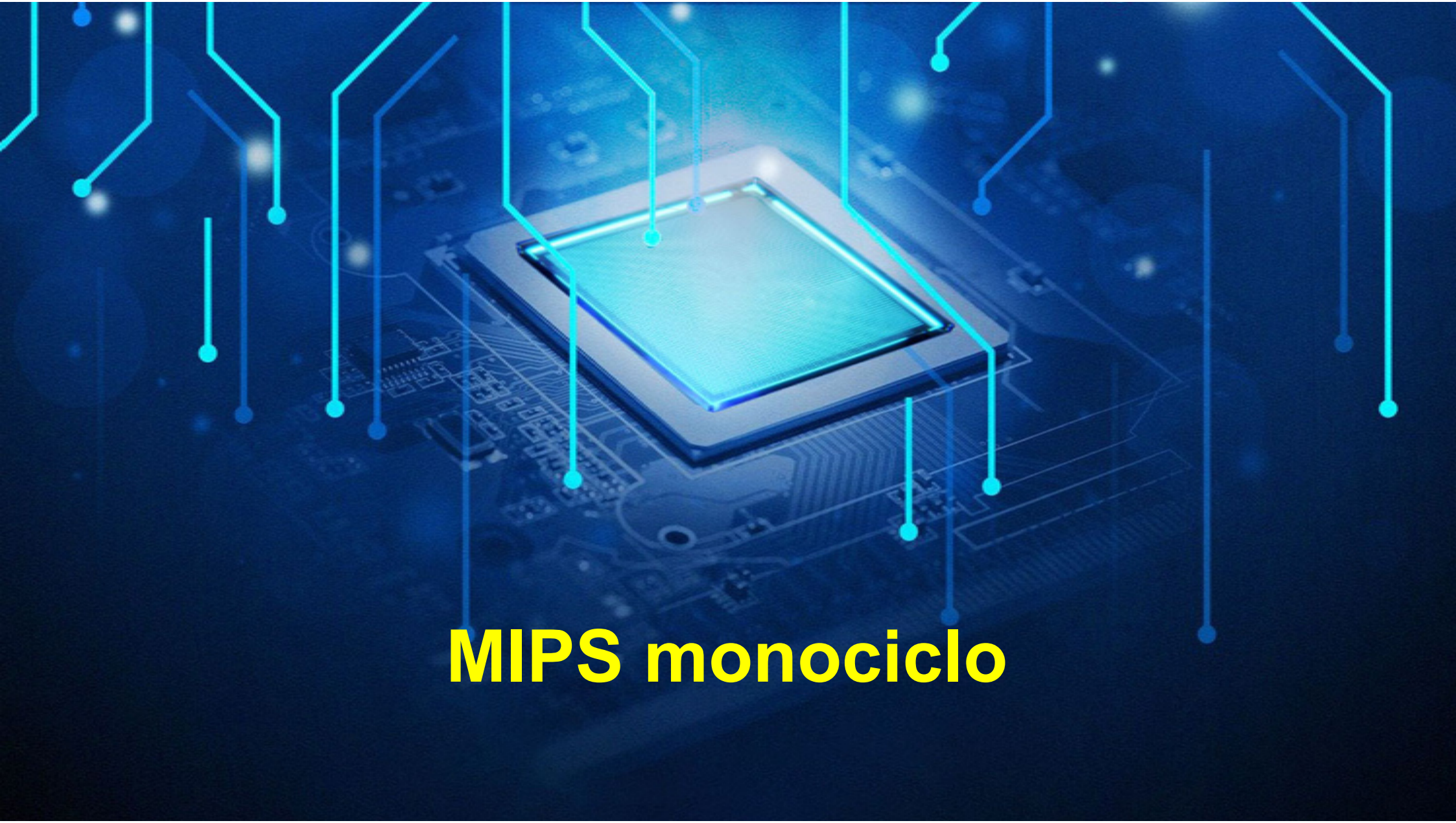
Formato delle Istruzioni

Il **formato C** è dedicato alle istruzioni riservate per il coprocessore matematico; il modulo in cui si svolgono operazioni aritmetiche tra numeri reali, rappresentati in virgola mobile secondo lo standard IEEE 754. Questo formato non è attualmente interessante per lo sviluppo della circuiteria della microarchitettura, perché il coprocessore è considerato come una periferica.

Formato delle istruzioni di tipo C (coprocessore) nel MIPS

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OPCODE						FORMAT					SOURCE 1					SOURCE 2					DESTINATION					FUNCTION					
0	1	0	0	x	x	b	b	b	b	b	b	b	b	b	b	b	b	b	b	b	s	s	s	s	s	f	f	f	f	f	f

Utilizzando questi quattro formati si ignorano aspetti come: specificare indirizzi di salto condizionato superiori a 2^{26}; gestire le chiamate a subroutine; usare costanti a 32bit. Benché siano elementi fondamentali, non risultano essere strettamente necessari per comprendere il principio di funzionamento della CPU monociclo del MIPS.

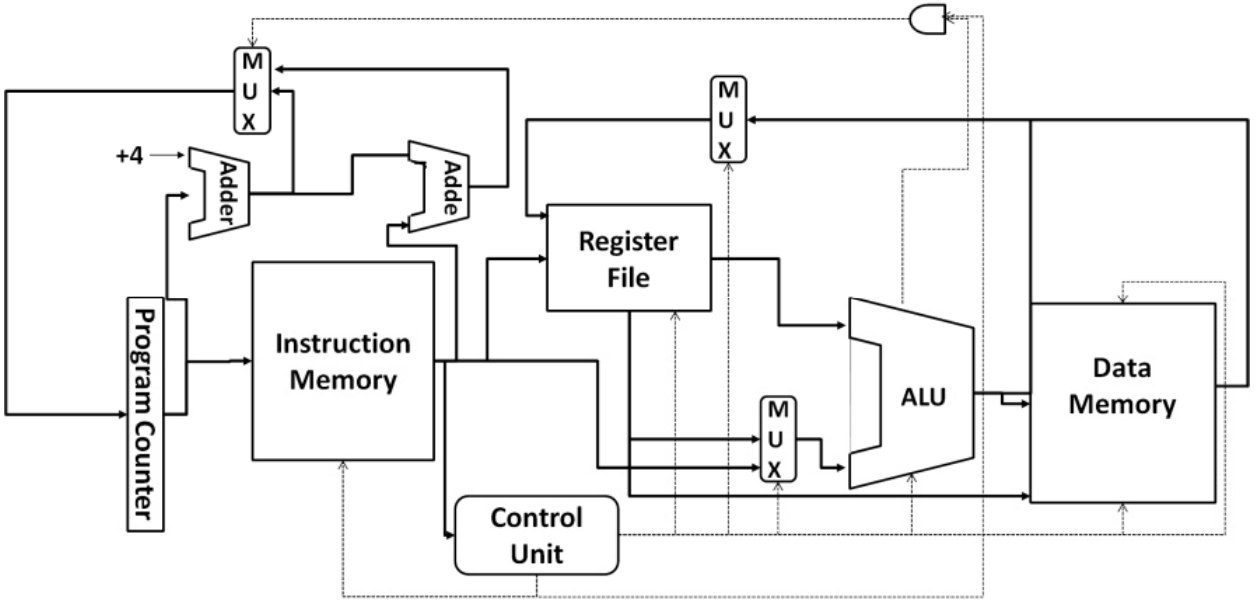

MIPS monociclo

MIPS Monociclo

Generalità

La circuiteria del MIPS, nella sua progettazione elementare, è in grado di elaborare in **un solo ciclo macchina ciascuna istruzione qualunque sia il formato (R, I, S)**.
Per questo si parla di **processore con datapath monociclo** o **elaborazione ad un colpo di clock**.

Il percorso dei dati (*datapath*) è composto da due elementi logici diversi (nel loro insieme chiamati Unità Funzionale): gli **elementi di stato** e gli **elementi combinatori**.
Il ciclo macchina, pertanto, è calibrato sull'istruzione che deve percorrere tutto il *datapath*.



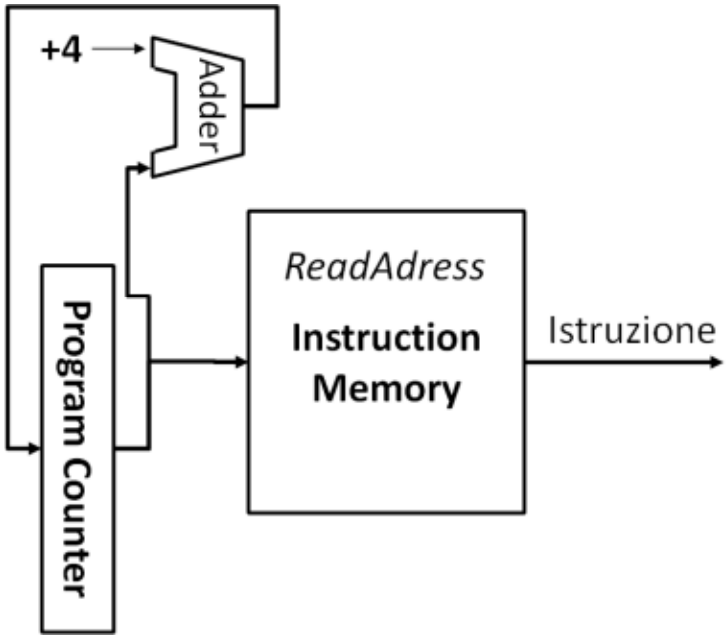
MIPS Monociclo

Prelievo istruzioni: Program Counter e Instruction Memory

Nel datapath del MIPS monociclo si sfrutta il **Contatore di Programma** per estrarre dalla Memoria delle Istruzioni quella da eseguire.
Il contatore di programma ha l'indirizzo alla locazione di memoria in cui è presente l'istruzione da dover elaborare

Nel contempo il Contatore di Programma è incrementato in modo tale che, se non ci sono salti, si possa prelevare l'istruzione contigua a quella elaborata

- L'incremento è di quattro unità perché la Memoria delle Istruzioni è fisicamente organizzata in byte e quindi parole di 32bit occupano quattro locazioni di memoria contigue ciascuna formata da celle di 8 bit (una circuiteria particolare, sfruttando un solo comando, consente di estrarre il contenuto delle quattro locazioni)



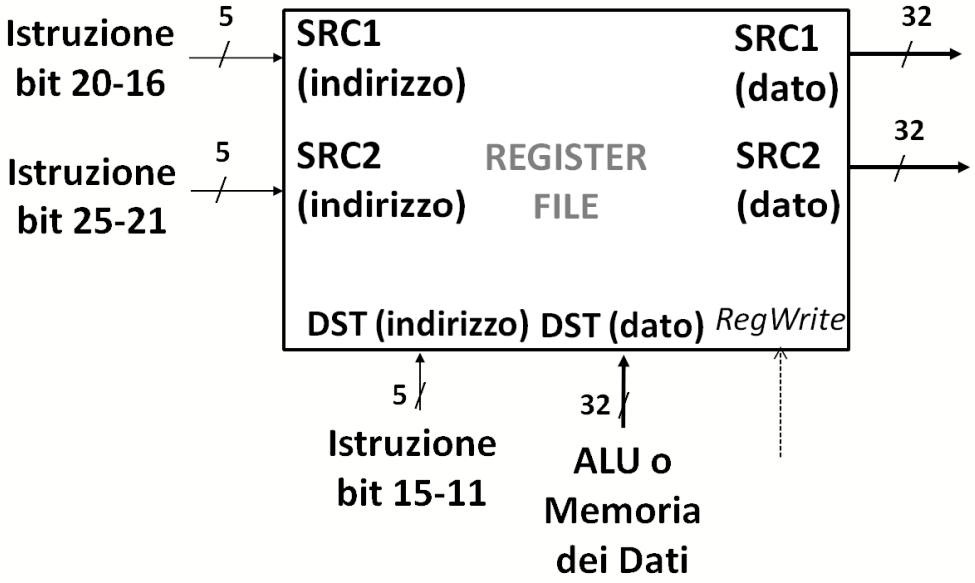
MIPS Monociclo

Banco dei registri

Nel datapath del MIPS sono presenti 32 registri (di cui 24 ad uso generale) che sono rappresentati da un'unità funzionale detta **Banco dei Registri** (*register file*).

- Di fatto è una memoria interna alla CPU, molto piccola, molto veloce e con una struttura di interconnessione che consente trasferimenti simultanei (es.: *mesh*)

Il Banco dei Registri comprende sia le unità di memoria (cioè i registri) sia la logica di controllo che permette di accedere a ciascuno dei registri, specificandone l'**indirizzo del registro** (numero o *tag*); il tipo di operazione, ovvero il **comando** di lettura o di scrittura; e di verificare se, e quando, il contenuto di un registro può essere modificato

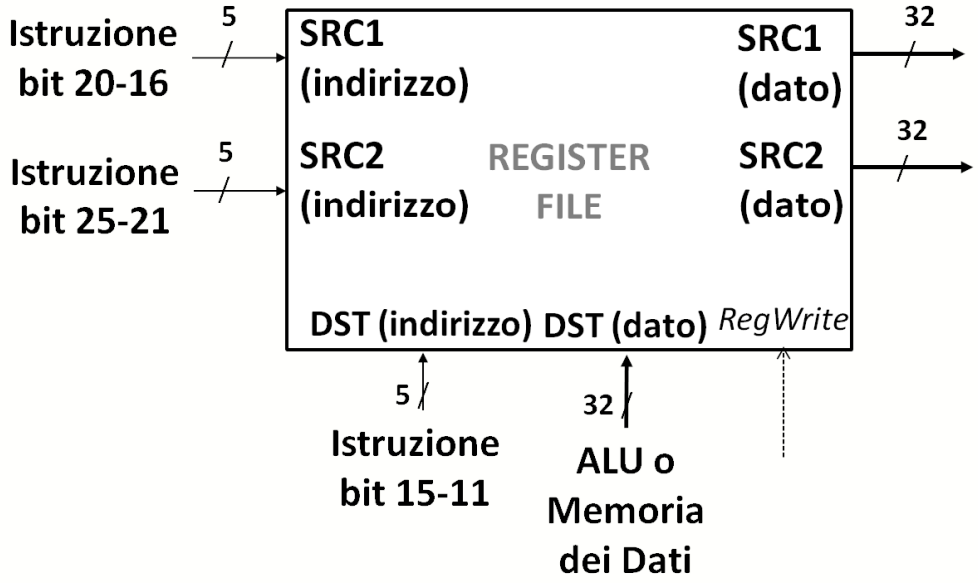


MIPS Monociclo

Comandi di gestione Banco dei Registri

Il comando RegWrite, generato dall'Unità di Controllo, quando è asserito, permette di **scrivere** l'operando proveniente dalla ALU (istruzione R) o dalla Memoria dei Dati (istruzione I) nel registro di destinazione, DST_{dato}, specificandone l'indirizzo, $DST_{indirizzo}$, ricavato analizzando i campi dell'istruzione in esecuzione

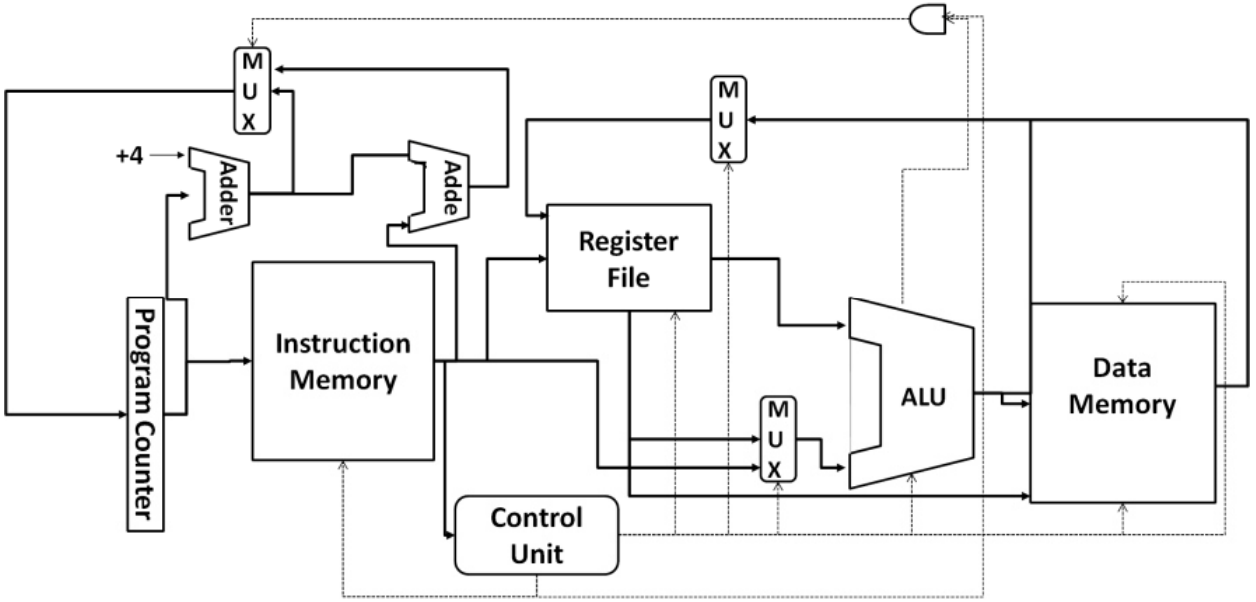
La **lettura di un registro**, invece, è immediata: in qualunque istante il Banco dei Registri fornisce in uscita su $SRC1_{dato}$ e $SRC2_{dato}$ il contenuto dei registri i cui tag sono specificati sugli ingressi $SRC1_{indirizzo}$ e $SRC2_{indirizzo}$.



MIPS Monociclo

ALU Control

Per realizzare l'**Unità di Controllo del MIPS con datapath monociclo** (in una forma semplificata e ridotta) bisogna dapprima definire i comandi associabili alla ALU, che riceve due operandi in ingresso e li combina usando la circuiteria della funzione logica-aritmetica specificata dal valore del segnale della sottorete di controllo che si occupa dell'Unità di Calcolo, cioè ***ALU Control***.



MIPS Monociclo

ALU Control

Pertanto **ALU Control** riceve in ingresso:

1. il campo FUNCTION presente nei sei bit meno significativi dell'istruzione in esecuzione
2. due bit di controllo ALUOp che sono già stati elaborati dall'Unità di Controllo e che stabiliscono se l'operazione che deve eseguire la ALU è una somma per calcolare l'indirizzo di trasferimento dalla Memoria Dati (load/store), una sottrazione per lo svolgimento di un salto condizionato o una operazione logica-matematica.

In uscita, qualora la ALU abbia 16 modi di funzionamento, ALU Control produce i comandi che sono codificati con quattro bit (vedere tabella a lato)

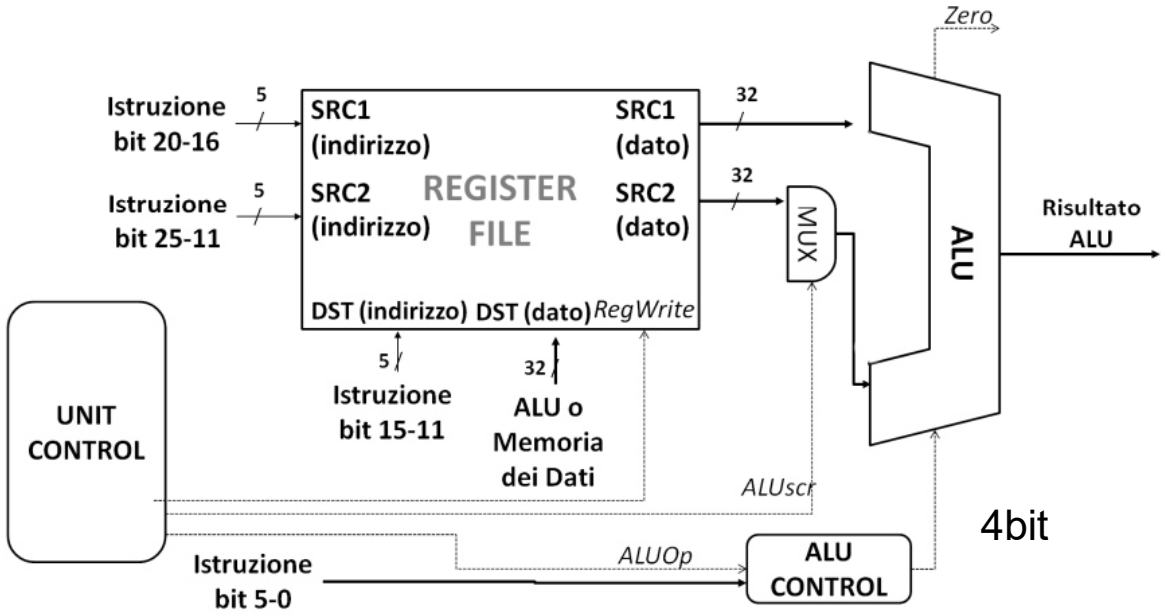
Sotto-unità di controllo della ALU (ALUControl) nel MIPS

Opcode	ALUop	Operazione	Function	Azione ALU	ALU Control (uscita)
Load	00	Lettura da memoria	xxxxxx	Somma	0010
Store	00	Scrittura in memoria	xxxxxx	Somma	0010
Branch equal	01	Salto in caso di operandi con ugual valore	xxxxxx	Sottrazione	0110
Tipo-R	10	Somma	100000	Somma	0010
Tipo-R	10	Sottrazione	100010	Sottrazione	0110
Tipo-R	10	And logico	100100	And	0000
Tipo-R	10	Or logico	100101	Or	0001
Tipo-R	10	Imposta ad 1 un registro se il primo registro sorgente è minore del secondo	101010	Set-On if less then	0111
…	…	…	…	…	…

MIPS Monociclo

ALUop

I due bit ALUop sono prodotti dall'Unità di Controllo sulla base del campo OPCODE, e insieme ai bit del campo FUNCTION costituiscono l'**input della ALU Control**, la cui **uscita sono i 4bit che selezionano il circuito operazionale** per ottenere il risultato



MIPS Monociclo

ALU Control

La **ALU Control** è realizzata con una rete combinatoria che soddisfa una **tabella di verità** (parzialmente rappresentata al lato destro)

La costruzione delle altre parti dell'Unità di Controllo procede in modo analogo a quanto visto, stabilendo per ogni componente (memorie, bus, periferiche) i relativi comandi.

Tabella della verità della ALU Control nel MIPS

ALUop1	ALUop2	F5	F4	F3	F2	F1	F0	ALUcontrol
0	0	x	x	x	x	x	x	0010
x	1	x	x	x	x	x	x	0110
1	x	x	x	0	0	0	0	0010
1	x	x	x	0	0	1	0	0110
1	x	x	x	0	1	0	0	0000
1	x	x	x	0	1	0	1	0001
1	x	x	x	1	0	1	0	0111
…	…	…	…	…	…	…	…	…

Le x indicano bit il cui valore è indifferente

MIPS Monociclo

Principali comandi Unità di Controllo MIPS

Per elaborare una istruzione, quindi, si devono **prelevare i 6bit ricevuti in ingresso dal campo OPCODE** (bit 31-26); **generare in uscita i comandi** per scrivere i registri (RegWrite); **provvedere alla lettura e alla scrittura della Memoria dei Dati** (MemRead e MemWrite); **controllare i multiplexer** (RegDst, ALUSrc, MemtoReg e PCsrc); nonché **amministrare la ALU**, cioè definire i valori di ALUop

Principali comandi dell'Unità di Controllo nel MIPS e loro significato		
Comando	**Attivo**	**Non attivo**
RegDst	Il numero del registro destinazione per Write Register proviene da SOURCE2 (bit 20-16)	Il numero del registro destinazione per Write Register proviene da DESTINATION (bit 15-11)
RegWrite	Nessuno	Nel registro specificato da Write Register è scritto il valore presente sull'ingresso Write Data
ALUSrc	Il secondo operando della ALU proviene dalla seconda uscita del Banco dei Registri	Il secondo operando della ALU sono i 16 bit meno significativi dell'istruzione
PCSrc	Il valore del Contatore di Programma è sostituito dall'uscita dell'ADDER che calcola PC+4	Il valore del Contatore di Programma è sostituito dall'uscita dell'ADDER che calcola la destinazione del salto
MemRead	Nessuno	Il contenuto della locazione di Memoria dei Dati con indirizzo ADDRESS è posto sull'uscita Read Data
MemWrite	Nessuno	Nella locazione di Memoria dei Dati con indirizzo ADDRESS è scritto il valore presente su Write Data
MemtoReg	il valore inviato all'ingresso di Write Data del Banco dei Registri proviene dalla ALU	Il valore inviato all'ingresso di Write Data del Banco dei Registri proviene dalla Memoria dei Dati

MIPS Monociclo

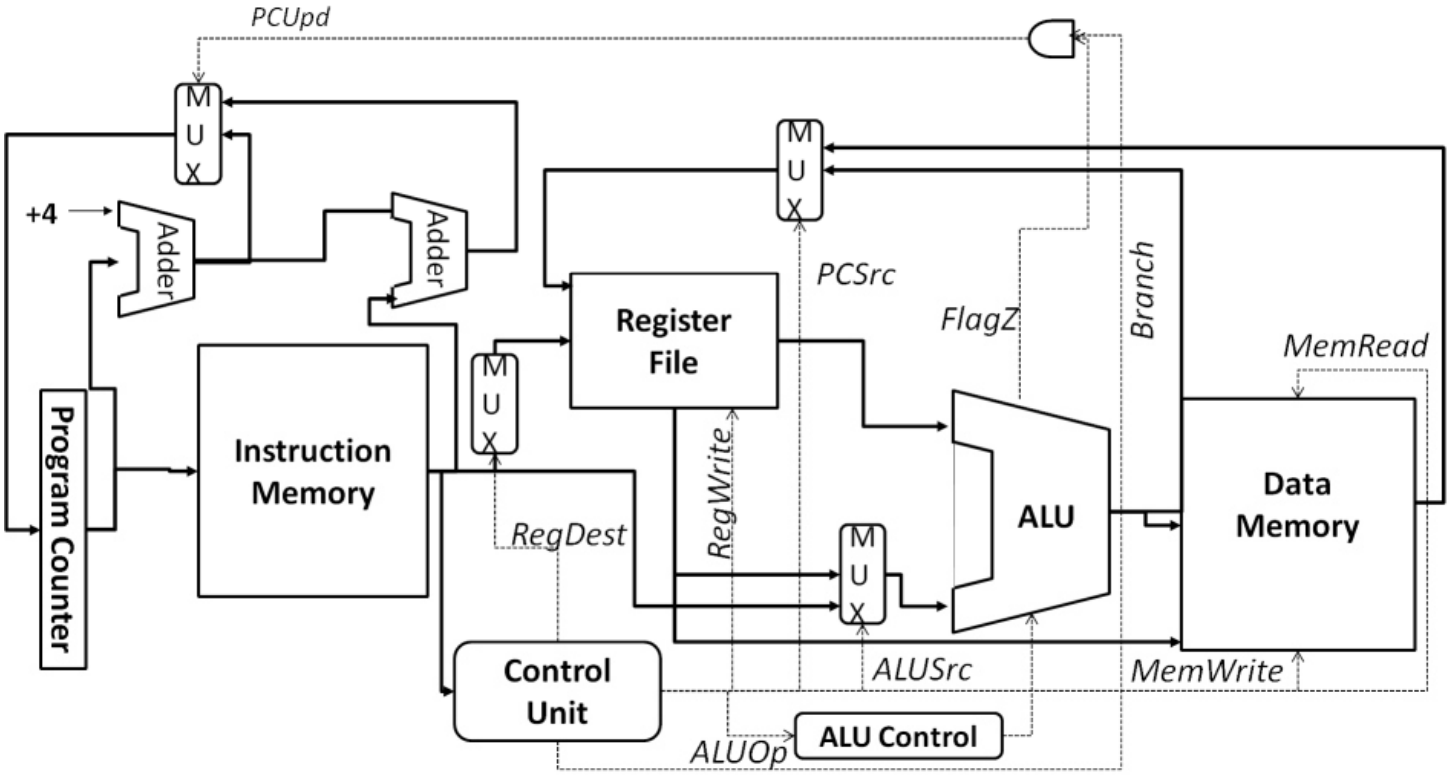
Unità di Controllo

L'**Unità di Controllo**, pertanto, è la **rete combinatoria** che realizza fisicamente la tabella di verità i cui valori di ingresso sono i sei bit del campo OPCODE di ciascuna istruzione e i valori di uscita sono i segnali di controllo del datapath MIPS (una parte è riportata in Tabella).

Tabella di verità delle istruzioni del MIPS di tipo-R (load, store e beq)					
	Segnale	**tipo-R**	**Load**	**store**	**Beq**
Input	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Output	RegDst	1	0	x	x
	ALUsrc	0	1	1	0
	MemtoReg	0	1	x	x
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

MIPS Monociclo

Da questi fattori si deriva la struttura dei comandi

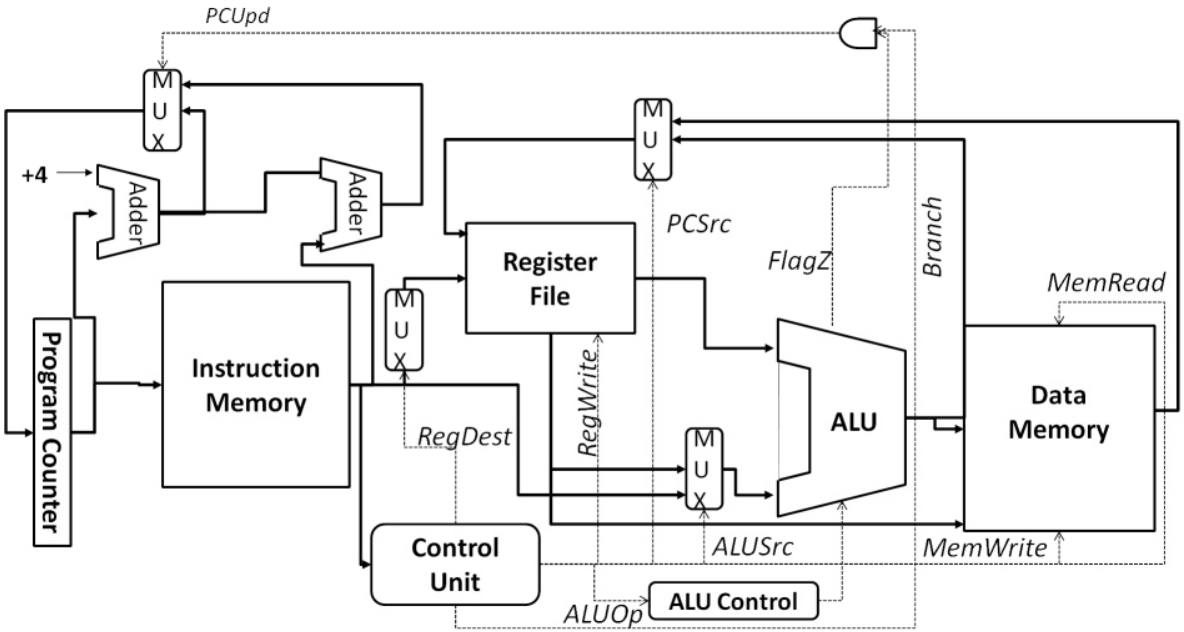


MIPS Monociclo

Esecuzione istruzione formato R

Una **istruzione in formato R**, ad esempio, è eseguita, in un clock compiendo i seguenti passi:

1. Il contenuto del Contatore di Programma è impiegato per indirizzare la Memoria delle Istruzioni che produce in uscita l'istruzione da eseguire.
2. Il campo OPCODE dell'istruzione è inviato all'Unità di Controllo, mentre i campi SOURCE1 e SOURCE2 sono usati per indirizzare il Banco dei Registri (in realtà l'istruzione va prima discriminata: bisogna riconoscerla come di tipo R)

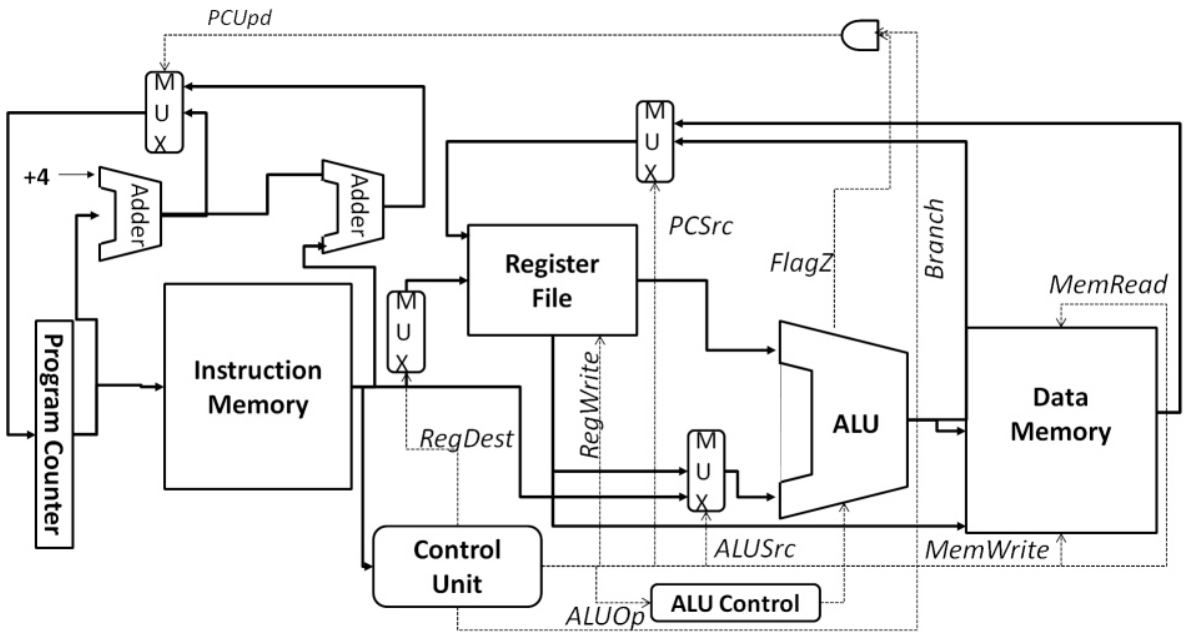


MIPS Monociclo

Esecuzione istruzione formato R

Una istruzione in formato R, ad esempio, è eseguita, in un clock compiendo i seguenti passi:

3. L'Unità di Controllo genera i comandi relativi ad una operazione di tipo-R (ALUOp=10). Questi sono inviati alla ALU Control insieme al campo FUNCTION dell'istruzione stabilendo l'effettiva operazione logico-aritmetica che deve eseguire l'Unità di Calcolo.
4. Nel frattempo, il Banco dei Registri restituisce gli operandi di SOURCE1 e SOURCE2, mentre il segnale ALUsrc=0 stabilisce che il secondo operando in ingresso alla ALU deve provenire dal Banco dei Registri.

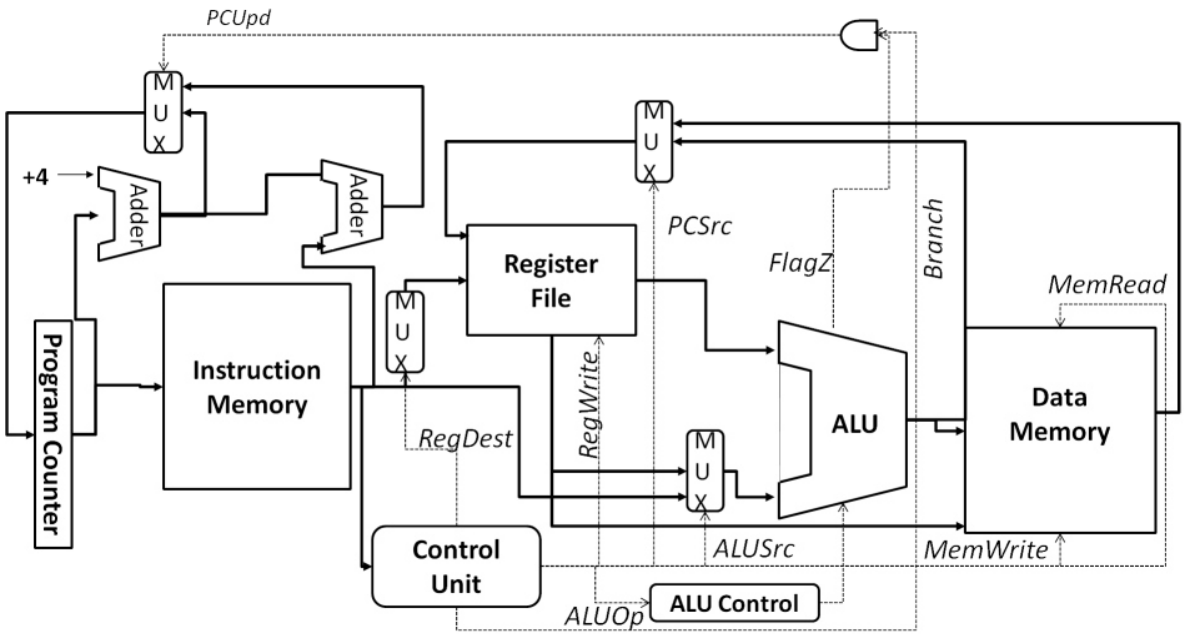


MIPS Monociclo

Esecuzione istruzione formato R

Una istruzione in formato R, ad esempio, è eseguita, in un clock compiendo i seguenti passi:

5. La ALU produce in uscita il risultato della computazione, che attraverso il segnale MemtoReg è presentato in ingresso al Banco dei Registri. Fino a questo punto non sono coinvolti elementi di stato: si sfruttano solamente reti combinatorie.
6. Il valore presente sull'ingresso WriteData del Banco dei Registro deve essere memorizzato nel registro il cui identificativo è specificato nel campo DESTINATION dell'istruzione, e la scrittura avviene nel momento in cui si presenta il fronte del clock a cui il banco dei registri è sensibile. Il comando RegWrite è asserito proprio per abilitare la scrittura del registro di destinazione specificato nell'istruzione.




FINE