



Architettura degli elaboratori Linguaggio Macchina e CPU

Prof. Alberto Borghese
Dipartimento di Informatica
Alberto.borghese@unimi.it

Università degli Studi di Milano

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



Sommario

- Linguaggio macchina
- Formato R
- Formato I
- Formato J
- Introduzione alla CPU



Linguaggio Macchina



- Le istruzioni assembler sono una **rappresentazione simbolica del linguaggio macchina** comprensibile dall'HW.
- Rappresentazione simbolica del linguaggio macchina
 - Più comprensibile del linguaggio macchina in quanto utilizza simboli invece che sequenze di bit
- Rispetto ai linguaggi ad alto livello, l'assembler fornisce limitate forme di controllo del flusso e non prevede articolate strutture dati
- Linguaggio usato come linguaggio target nella fase di compilazione di un programma scritto in un linguaggio ad alto livello (es: C, Pascal, ecc.)
- Vero e proprio linguaggio di programmazione che fornisce la visibilità diretta sull'hardware.

**Codice in
linguaggio ad alto
livello (C)**

```
a = a + c
b = b + a
var = m[a]
```

↓

Codice Assembler

```
add $t0, $s0, $s1
add $s2, $s2, $t0
multi $t1, $s4, 4
add $t2, $s5, $t1
lw $s3, 0($t2)
```


↓

**Codice in
linguaggio
macchina**


```
011100010101010
000110101000111
000010000010000
001000100010000
```

mimi.it\

A.A. 2021-2022
3/58



Codifica delle istruzioni



- Tutte le istruzioni MIPS hanno la **stessa** dimensione (**32 bit**) – **Architettura RISC**.
- 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 prevalentemente 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 con una costante.
 - **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 / costante		
J	op	Indirizzo / costante				

A.A. 2021-2022
4/58
<http://borghese.di.unimi.it>



Sommario



- Linguaggio macchina
- **Formato R**
- Formato I
- Formato J
- Introduzione alla CPU

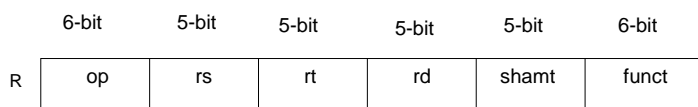


Formato delle istruzioni di tipo R



Contiene:

- Un codice operativo su 6 bit
- Un registro source, rs, su 5 bit (32 registri)
- Un registro target, rt, su 5 bit (32 registri)
- Un registro destinazione, rd, su 5 bit (32 registri)
- Un numero di posizioni di shift (shift amount, shamt), su 5 bit
- Un codice di funzione (cf. selettore ALU), su 6 bit



Istruzioni di tipo R: esempio

Nome	Numero	Utilizzo
\$zero	0	costante zero
\$at	1	riservato per l'assemblatore
\$v0-\$v1	2-3	valori di ritorno di una procedura
\$a0-\$a3	4-7	argomenti di una procedura
\$t0-\$t7	8-15	registri temporanei (non salvati)
\$s0-\$s7	16-23	registri salvati
\$t8-\$t9	24-25	registri temporanei (non salvati)
\$k0-\$k1	26-27	gestione delle eccezioni
\$gp	28	puntatore alla global area (dati)
\$sp	29	stack pointer
\$s8	30	registro salvato (fp)
\$ra	31	indirizzo di ritorno

add \$t8, \$s1, \$s2

0x0232A020

A.A. 2021-2022 7/58 <http://borghese.di.unimi.it/>

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
add \$t8, \$s1, \$s2	000000	10001	10010	11000	00000	100000

Nome campo	op	rs	rt	rd	shamt	funct
Dimensione	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
sub \$t8, \$s1, \$s2	000000	10001	10010	11000	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, 3	000000	X	10010	10001	00011	000000

$s1 = s2 * 2^3$ Se $s2$ contiene 20 (0000.....0010100) => $s1$ conterrà = $20 * 2^3 = 160$ (0000...0010100000)

Nome campo	op	rs	rt	rd	shamt	funct
Dimensione	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
srl \$s1, \$s2, 3	000000	X	10010	10001	00011	000010

$s1 = s2 * 2^{-3}$

A.A. 2021-2022 8/58 <http://borghese.di.unimi.it/>



Sommario



- Linguaggio macchina
- Formato R
- **Formato I**
- Formato J
- Introduzione alla CPU



Formato istruzioni di tipo I



I	op	rs	rt	costante / indirizzo
	6 bit	5 bit	5 bit	16 bit

- 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 / indirizzo**. Nel caso di una lw riporta lo spiazamento (offset) nel segmento dati, nel caso di una beq riporta lo spiazamento nel segmento testo.

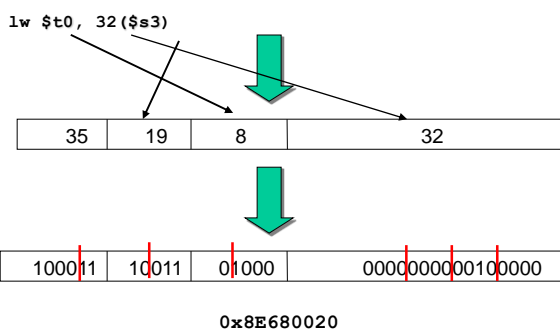


Istruzioni di tipo I: accesso a memoria



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

$$\text{indirizzo} = \text{indirizzo_base} + \text{offset} (= \text{costante})$$

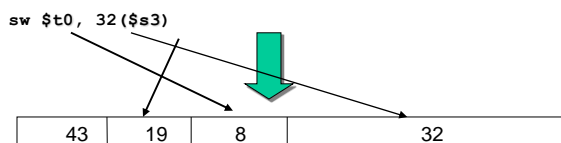


Istruzioni di tipo I: accesso memoria



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

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



Differenza di 1 bit (distanza di Manhattan pari a 1) ->
cambia la direzione del trasferimento con la memoria.



Istruzioni di tipo I: aritmetico-logiche



Nome campo	op	rs	rt	costante
Dimensione	6-bit	5-bit	5-bit	16-bit
<code>addi \$t0, \$s3, 64</code>	001000	10011	01000	0000 0000 0100 0000

Nome campo	op	rs	rt	costante
Dimensione	6-bit	5-bit	5-bit	16-bit
<code>andi \$t0, \$s3, 64</code>	001100	10011	01000	0000 0000 0100 0000

Nome campo	op	rs	rt	costante
Dimensione	6-bit	5-bit	5-bit	16-bit
<code>slti \$t0, \$s2, 8</code>	001010	10010	01000	0000 0000 0000 1000

\$t0 = 1 if \$s2 < 8

NB ruolo del registro target: nelle istruzioni di `addi` e di `lw` rappresenta **dove nel register file vado a scrivere**. Nelle istruzioni di tipo R e di `sw` contiene **uno dei dati sorgente**.

A.A. 2021-2022

13/58

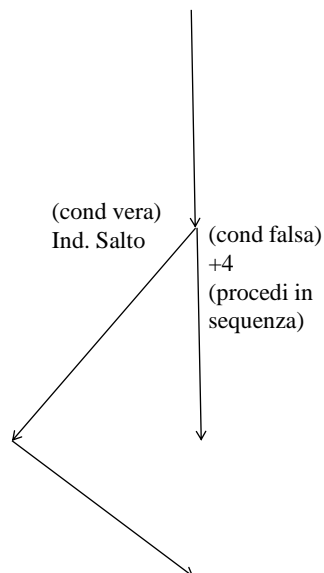
<http://borghese.di.unimi.it/>



Istruzioni di tipo I: salto condizionato



- Salti condizionati relativi:
 - `beq r1, r2, L1` (*branch on equal*)
 - `bne r1, r2, L1` (*branch on not equal*)
- Salti condizionati assoluti:
 - 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)** avviene a partire dal Program Counter (PC).
 - Indirizzamento del tipo Base (PC) + Spiazzamento.



A.A. 2021-2022

14/58

<http://borghese.di.unimi.it/>



Allargamento dello spazio di indirizzamento relativo



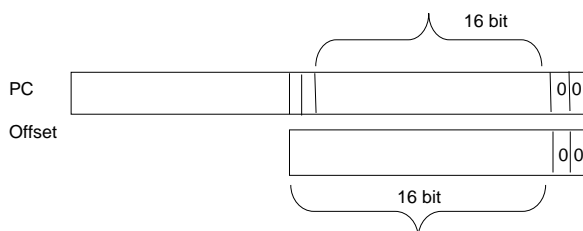
0000	0	0
0100	1	4
1000	2	8
1100	3	12

Gli offset in Byte avranno sempre **00** come ultimi 2 bit perché gli indirizzi sono multipli di 4.

Calcolo lo spiazamento in numero di parole invece che di Byte.

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

Moltiplicare per 4 vuol dire operare uno shift a sinistra di due posizioni dell'offset



La costante su 16 bit, contenuta nel codice, rappresenta l'offset in termini di numero di istruzioni

A.A. 2021-2022

15/58

<http://borghese.di.unimi.it/>



Calcolo dell'indirizzo di salto



```

Repeat
    0x0000      addi $t1, $zero, 10  # N=10
    0x0004      addi $t0,$zero,0    # i =0;
{   ....      0x0008  loop:  addi $t0, $t0,1  # i++
    0x000C      ....
    0x0020      bne $t0, $t1, loop
} until (i == N)



```

Quanto vale il campo costante da inserire nella stringa dell'istruzione bne?

A.A. 2021-2022

16/58

<http://borghese.di.unimi.it/>

Calcolo dell'indirizzo di salto

Ind Dec	Ind Exadec	Istruzione	# istruzione
000	0x0000	addi \$t1, \$zero, 10 # N=10	1
004	0x0004	addi \$t0,\$zero,0 # i =0;	2
008	0x0008	loop: addi \$t0, \$t0,1 # i++	3
012	0x000C	4
032	0x0024	bne \$t0, \$t1, loop	
036	0x0028		9

L'indirizzo di destinazione è 0x8 (indirizzo dell'etichetta loop)

Lo spiazzamento del salto in byte è pari a: $(0x8 - 0x24) = (8-36) = -28$ Byte
 Lo spiazzamento del salto in numero di istruzioni è pari a -7 istruzioni



Prova: Indirizzo di salto = Indirizzo PC+4 + Offset (#istruzioni) * 4
 loop = 8 → costante = 36 + (-7) * 4

5	8	9	-7 = 1111 1111 1111 1001
---	---	---	--------------------------

A.A. 2021-2022

17/58

<http://borghese.di.unimi.it/>

Istruzioni di tipo I – Branch (esempi)

Nome campo	op	rs	rt	indirizzo
Dimensione	6-bit	5-bit	5-bit	16-bit
beq \$s1, \$s2, L1	000100	10001	10010	0000 0000 0001 1001

L1 = PC+4 + 100 (byte) Codifica su 18 bit: 0000 0000 0001 1001(00) in binario.

Nome campo	op	rs	rt	indirizzo
Dimensione	6-bit	5-bit	5-bit	16-bit
beq \$s1, \$s2, L1	000100	10001	10010	1111 1111 1110 0111

L1 = PC+4 -100 (byte) Codifica su 18 bit: 1111 1111 1110 0111(00) in binario.

A.A. 2021-2022

18/58

<http://borghese.di.unimi.it/>



Osservazioni



Nome campo	op	rs	rt	Indirizzo			
Dimensione	6-bit	5-bit	5-bit	16-bit			
beq \$s1, \$s2, L1	000100	10001	10010	0000	0000	0001	1000

Nome campo	op	rs	rt	Indirizzo			
Dimensione	6-bit	5-bit	5-bit	16-bit			
lw \$s2, 24(\$s1)	100011	10001	10010	0000	0000	0001	1000

Nome campo	op	rs	rt	costante			
Dimensione	6-bit	5-bit	5-bit	16-bit			
addi \$s2, \$s1, 24	001000	10001	10010	0000	0000	0001	1000

Istruzioni molto diverse possono distare pochi bit una dall'altra.



Formato R ed operazioni logico-matematiche



Non tutte le operazioni logico-matematico, sono di tipo R.



Le operazioni logico-matematiche di tipo R hanno codice operativo 0.

Non tutte le operazioni con codice operativo 0 sono logico-matematiche (ad esempio ci sono le istruzioni di *jr, syscall...*).

Occorre distinguere il funzionamento dell'istruzione elementare dalla sua codifica.

- Codifiche simili (e.g. Tipo R) possono essere condivise da istruzioni di tipo diverso (e.g. aritmetico-logiche, salto).
- Codifiche diverse (e.g. Tipo I e Tipo R) possono essere condivise da istruzioni dello stesso tipo (e.g. add ed addi)

Non c'è corrispondenza 1 a 1, tra tipi strutturali e tipi funzionali.



Sommarrio

- Linguaggio macchina
- Formato R
- Formato I
- **Formato J**
- Introduzione alla CPU

A.A. 2021-2022

21/58

<http://borghese.di.unimi.it/>

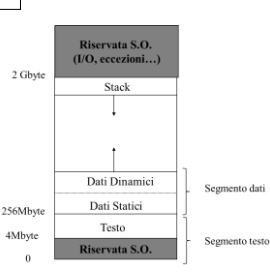
Formato istruzioni di tipo J

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

j	indirizzo
op	indirizzo

6 bit
26 bit

- 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 = 256 Mbyte (segmento testo).



PC

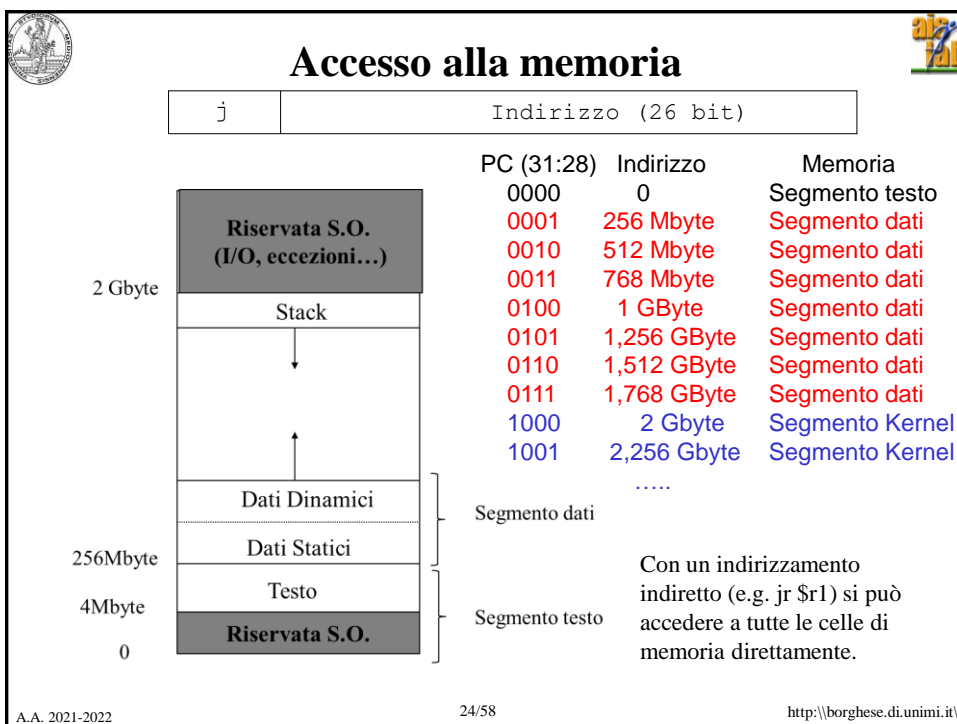
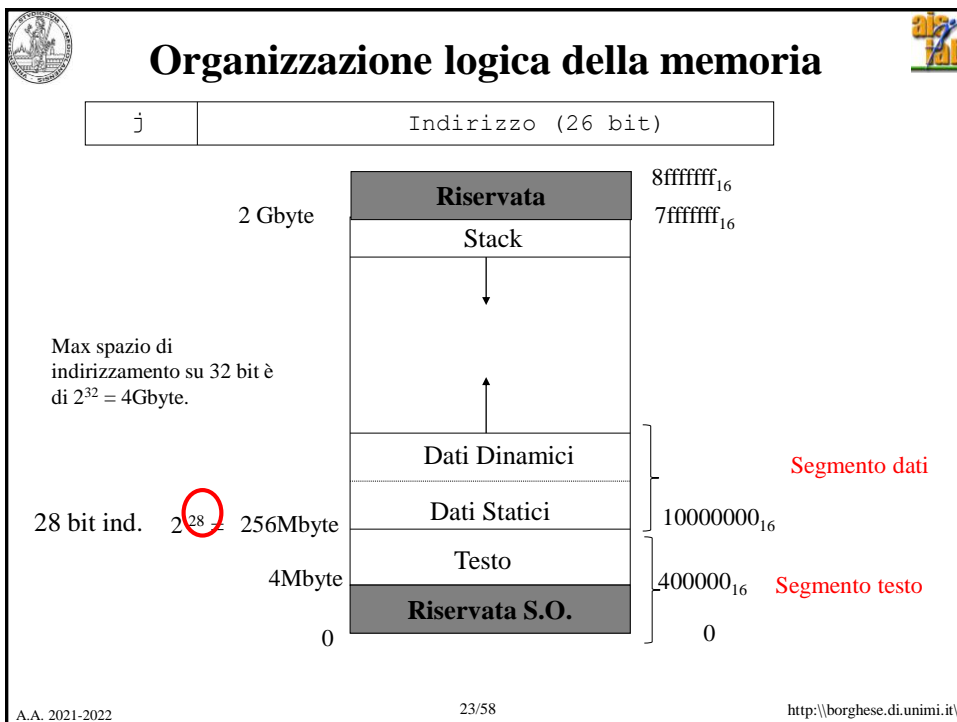
4 bit (invariati)	26 bit	2 bit

$0 \leq \text{indirizzo} < 2^{28} = 256 \text{ Mbyte (segmento testo!)}$

A.A. 2021-2022

22/58

<http://borghese.di.unimi.it/>





Codifica delle istruzioni



- Tutte le istruzioni MIPS hanno la **stessa dimensione (32 bit)** – **Architettura RISC.**
- 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 / costante		
J	op	Indirizzo / costante				



Formati e tipi di istruzioni



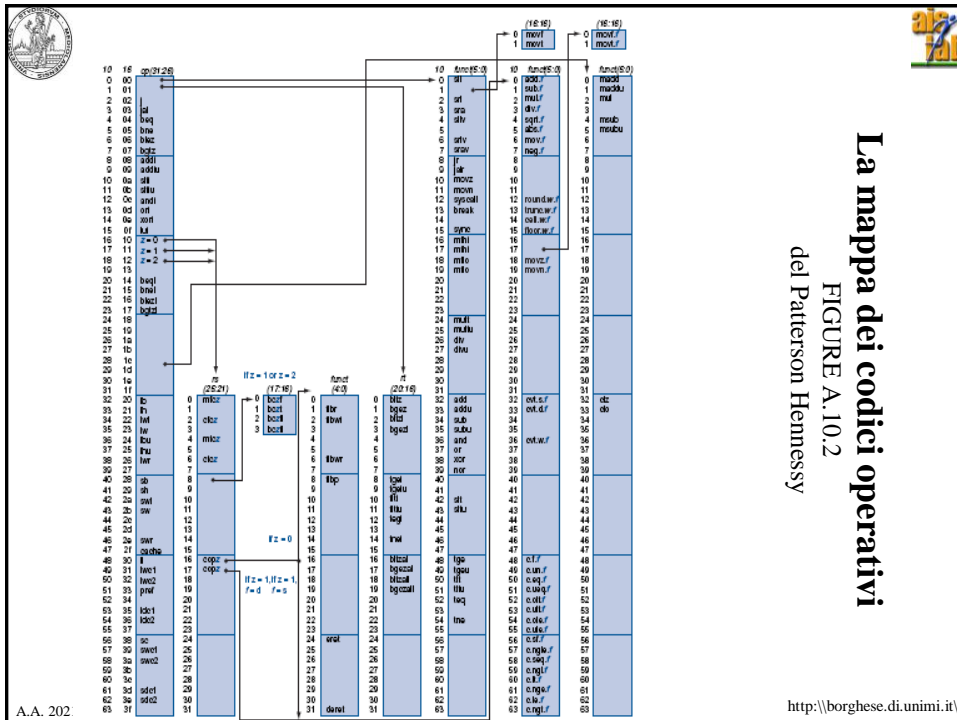
Tipi di istruzioni di un'ISA

- Istruzioni aritmetiche
- Istruzioni di accesso a memoria
- Istruzioni di controllo di flusso
- (Istruzioni di I/O)

Formati

- R - Registro
- I - Immediato
- J - Jump (salto)

Non c'è corrispondenza 1 a 1 tra tipi di istruzioni e formati



Sommarrio

- Linguaggio macchina
- Formato R
- Formato I
- Formato J
- **Introduzione alla CPU**

A.A. 2021-2022 28/58 <http://borghese.di.unimi.it/>



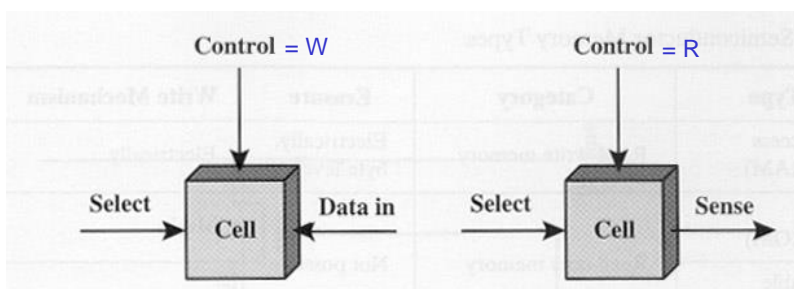
Cella di memoria



La memoria è suddivisa in celle, ciascuna delle quali assume un valore binario stabile.

Si può scrivere il valore 0/1 in una cella.

Si può leggere il valore di ciascuna cella.



Control (lettura – scrittura)

Select (selezione)

Data in oppure Data out (sense)

Base

A.A. 2021-2022

29/58

<http://borghese.di.unimi.it/>



Latch sincrono come elemento di memoria

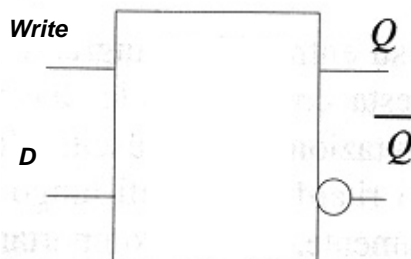
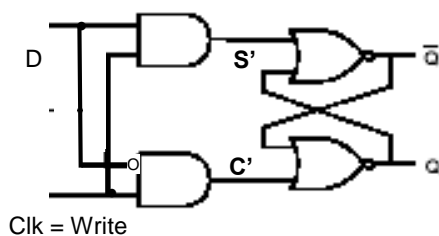


L'ingresso "clock" del bistabile viene utilizzato come segnale di write

E' trasparente quando Write = 1

Se Write = 1 $Q_{t+1} = D$


Se Write = 0 $Q_{t+1} = Q_t$




A.A. 2021-2022

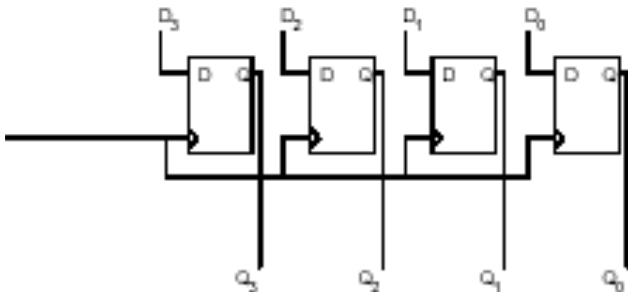
30/58

<http://borghese.di.unimi.it/>



Registri





Un registro a 4 bit.
 Memorizza 4 bit.
 Tutti gli elementi di memoria ricevono lo stesso segnale di write


NB Non è un registro a scorrimento (shift register!)

Latch o flip-flop di tipo D


A.A. 2021-2022

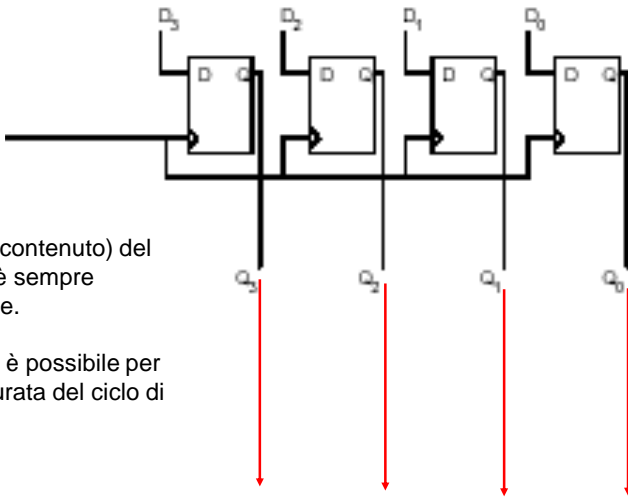
31/58

<http://borghese.di.unimi.it/>



Letture di un registro





Lo stato (contenuto) del bistabile è sempre disponibile.

La lettura è possibile per tutta la durata del ciclo di clock.

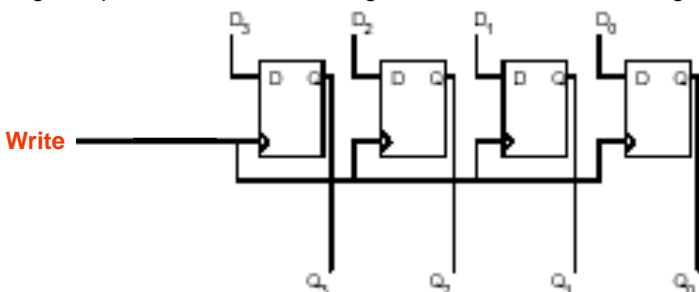
A.A. 2021-2022

32/58

<http://borghese.di.unimi.it/>

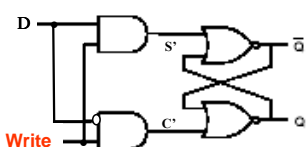
Scrittura di un registro

Ad ogni colpo di clock lo stato del registro assume il valore dell'ingresso dati.




Cosa occorre modificare perchè il registro venga scritto quando serve?
 Introdurre una sorta di *"apertura del cancello (chiusura circuito)"*.
 Può essere sincronizzata o meno con il clock.

3
 Il segnale di write apre il passaggio al contenuto di D attraverso il latch. Quando il segnale di Write è a zero, lo stato non varia.

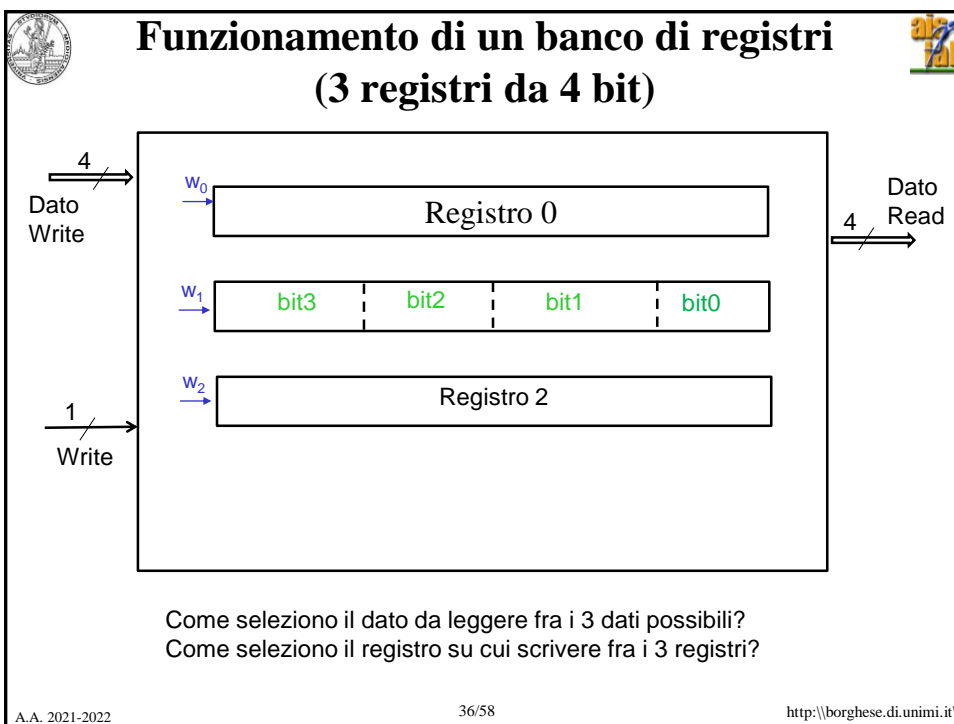
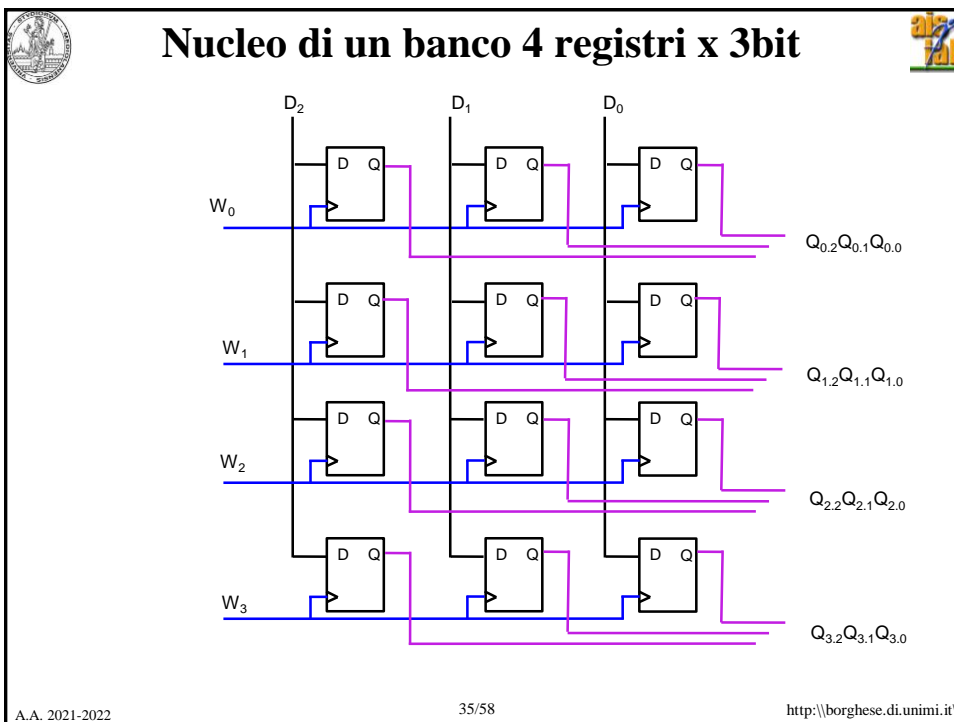


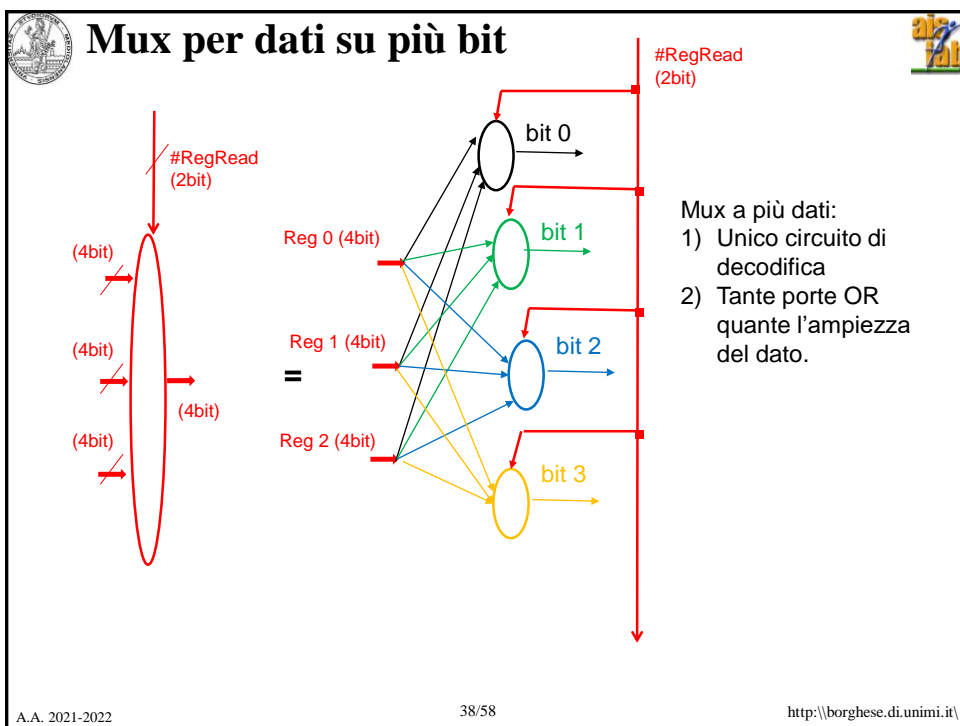
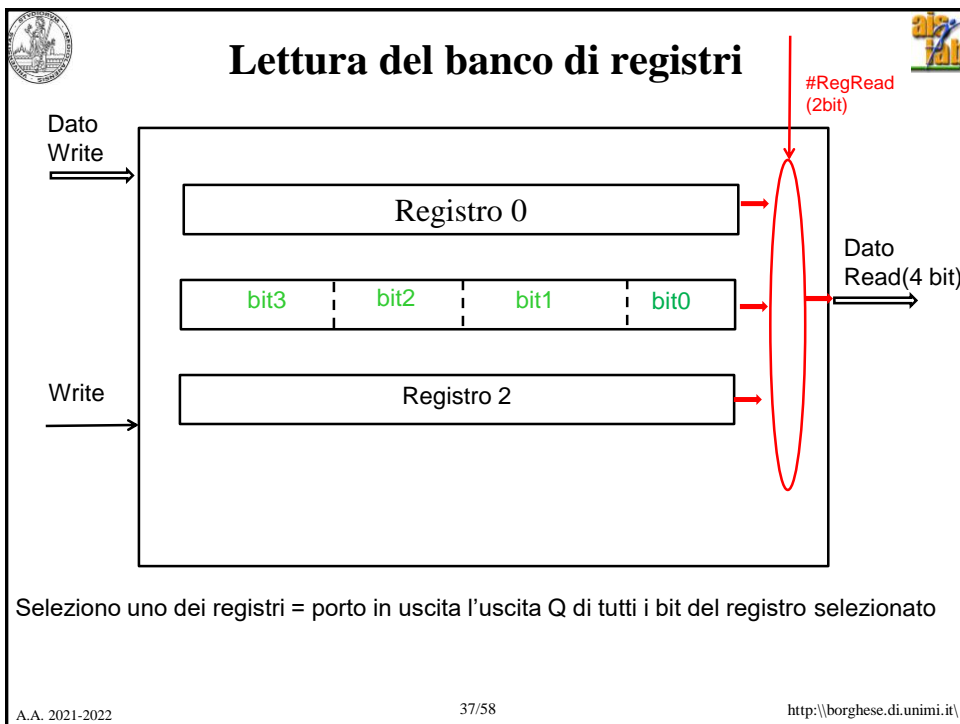
A.A. 2021-2022 33/58

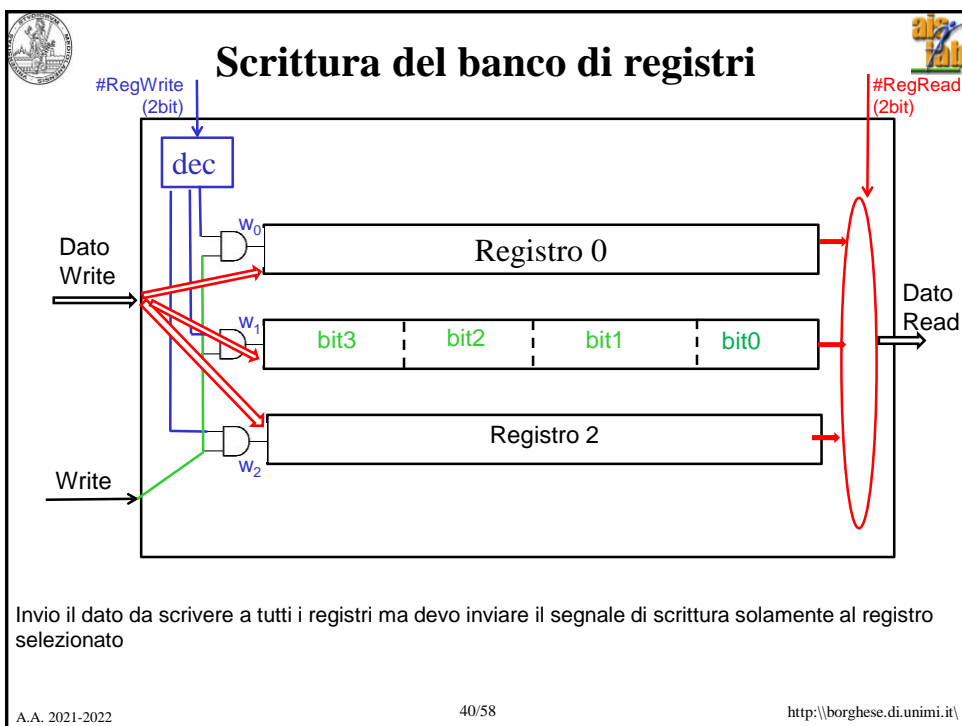
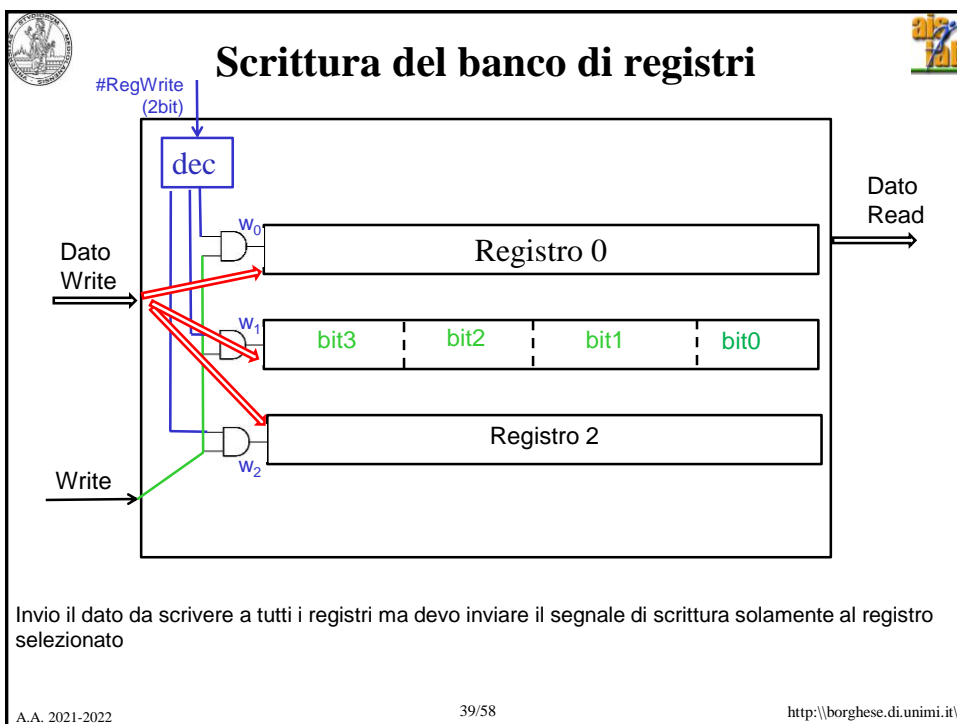
Registro

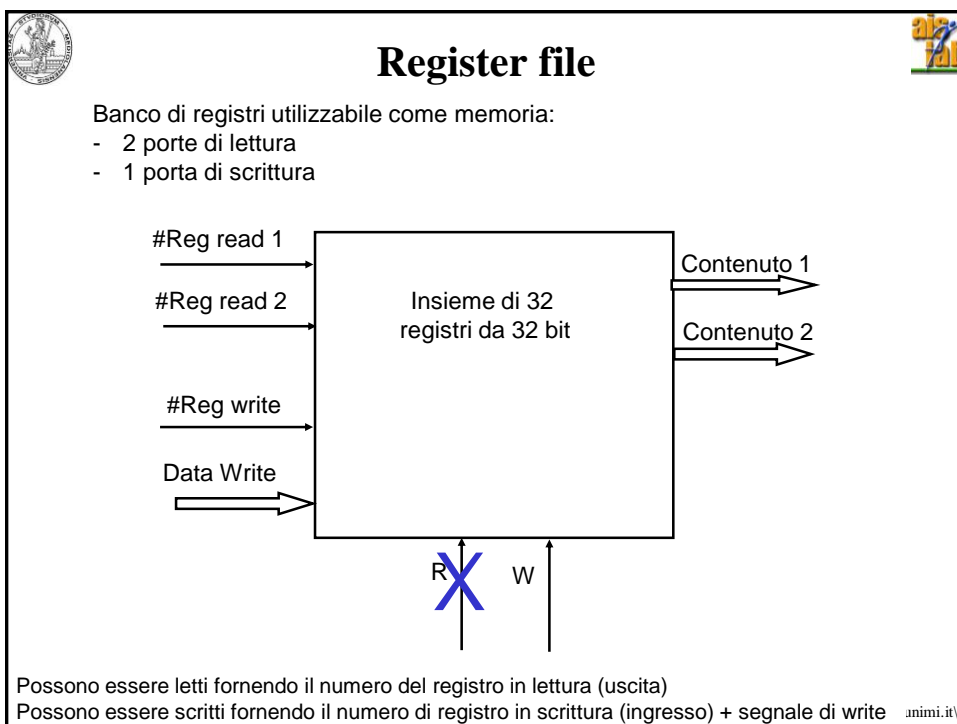
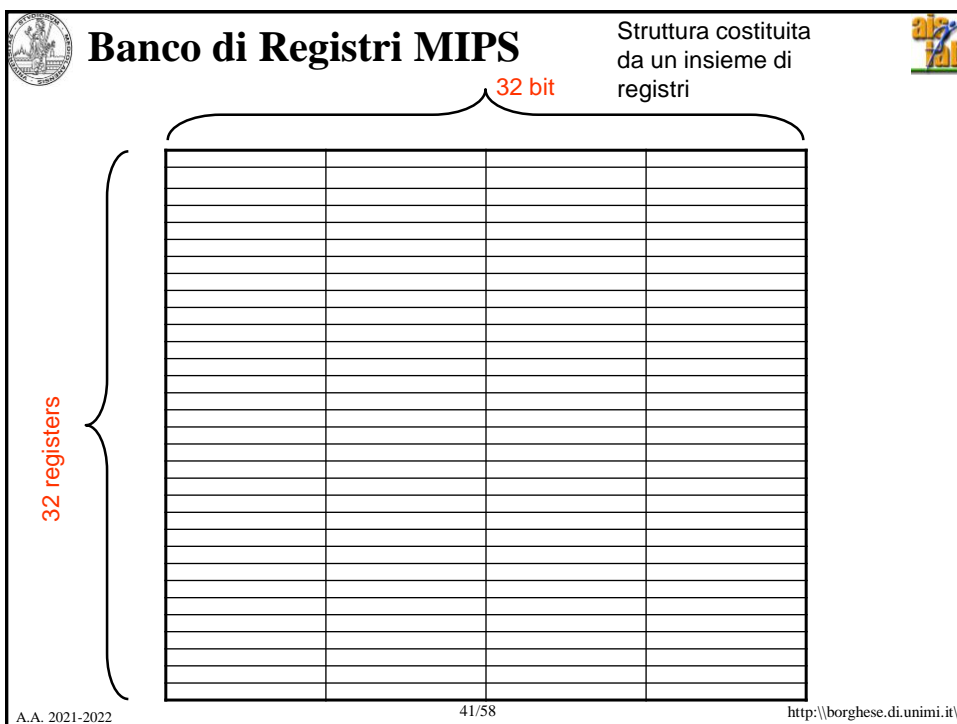


A.A. 2021-2022 34/58 <http://borghese.di.unimi.it/>









Gestione del register file

La lettura **non modifica** il contenuto di un registro (collego l'uscita Slave con il circuito combinatorio).

La scrittura invece **modifica** il contenuto del registro. Occorre il segnale Write esplicito.

$\#bit_indirizzamento = \log_2 \#bit$

A.A. 2021-2022 43/58 <http://borghese.di.unimi.it/>

Porta di lettura del register file

`add $t0, $t1, $t2`

Un mux per ogni porta di lettura.

Ciascun Mux ha la complessità di 32 mux, uno per ogni bit.

Numero del registro letto 1 $5/$
\$t1 = 9

Numero del registro letto 2 $5/$
\$t2 = 10

A.A. 2021-2022 44/58 <http://borghese.di.unimi.it/>

Porta di scrittura del register file

Ingresso C del latch dei registri:
 Decodificatore per indirizzare il registro
 AND
 Comando W

Ingresso D del latch dei registri:
 Bit dato corrispondente.

A.A. 2021-2022 45/58 <http://borghese.di.unimi.it/>

Register file del MIPS

Numero del registro letto 1

Numero del registro letto 2

RegWrite

#Reg Write

Dato

A.A. 2021-2022 46/58 <http://borghese.di.unimi.it/>



Obiettivo

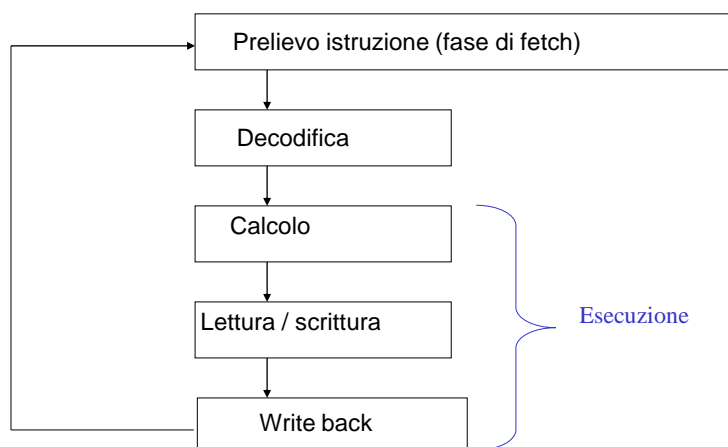


Costruzione di una CPU completa che sia in grado di eseguire:

- Istruzioni logico-matematiche di tipo R (e.g. add, sub, and...). e.g. add \$t0, \$t1, \$t2).
- Istruzioni logico-matematiche di tipo I (e.g. addi, ori...) e.g. addi \$t0, \$t1, 24.
- Accesso alla memoria in lettura (lw) o scrittura (sw). e.g. lw \$t0, 24(\$t1)
- Istruzioni di salto condizionato (branch). e.g. beq \$t0, \$t1, etichetta
- Istruzioni di salto incondizionato (jump). e.g. j etichetta



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*), sommatore ausiliari, ecc.;
- **Unità di controllo**. Controlla il flusso e determina le operazioni di ciascun blocco.

MEMORIA PRINCIPALE




Codifica delle istruzioni




- Tutte le istruzioni MIPS hanno la **stessa** dimensione (**32 bit**) – **Architettura RISC**.
- 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 prevalentemente 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 con una costante.
 - **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 / costante		
J	op	Indirizzo / costante				



Istruzioni




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


A.A. 2021-2022

51/58

<http://borghese.di.unimi.it/>



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 (segmento testo).
- Il segnale di controllo per la lettura (READ) viene inviato alla memoria.
- Trascorso il tempo necessario alla lettura dalla memoria, la parola indirizzata (in questo caso la prima istruzione del programma) si troverà nel registro IR.
- Il contenuto del PC viene incrementato in modo da puntare all'istruzione successiva.

A.A. 2021-2022

52/58

<http://borghese.di.unimi.it/>

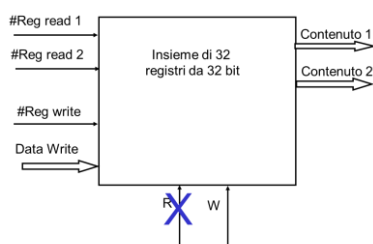


Decodifica dell'istruzione



- L'istruzione contenuta nel registro IR viene decodificata per essere eseguita. Alla fase di decodifica corrisponde la predisposizione della CPU (apertura delle vie di comunicazione 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 LOAD/STORE: Le istruzioni di caricamento dalla memoria sono separate da quelle aritmetico/logiche.

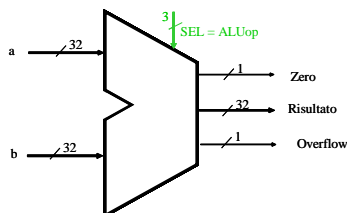


Calcolo dell'istruzione (execute - calcolo)



Viene selezionata all'interno della ALU l'operazione prevista dall'istruzione e determinata in fase di decodifica dell'istruzione.

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 dati



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

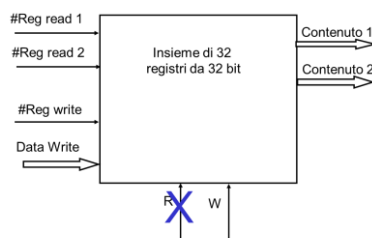
Nel caso particolare di Architetture LOAD/STORE, quali MIPS, le istruzioni di caricamento dalla memoria sono separate da quelle aritmetico/logiche. Se effettuato una Lettura / Scrittura, **non** eseguo operazioni aritmetico logiche sui dati.



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.





Come funziona una CPU?



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

 - n L'UC 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.

- n 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, <...).

- n Accesso alla memoria (se richiesto).

- n Scrittura del risultato nel register file (se richiesto).



Sommario



- Linguaggio macchina
- Formato R
- Formato I
- Formato J
- Introduzione alla CPU