


Linguaggio macchina e register file

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

Università degli Studi di Milano

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


Il linguaggio macchina

- I diversi tipi di istruzioni: Istruzioni di tipo I
- I diversi tipi di istruzioni: Istruzioni di tipo J
- Register file

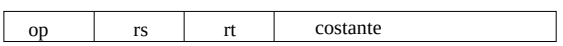
Formati e tipi di istruzioni

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

Formato istruzioni di tipo I

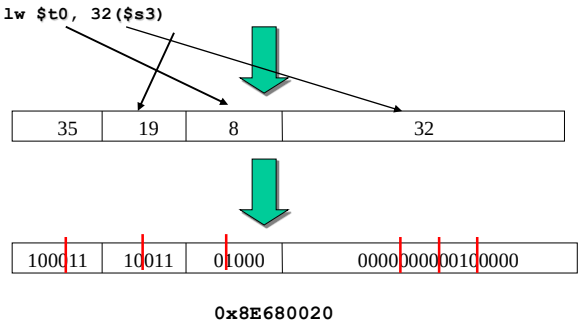


6 bit 5 bit 5 bit 16 bit

- 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`**. Nel caso di una lw riporta lo spiazzamento (offset).

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: indirizzo = indirizzo_base + offset (= costante)



`0x8E680020`

Istruzioni di tipo I: accesso memoria

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

Versione I di istruzioni aritmetico-logiche

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

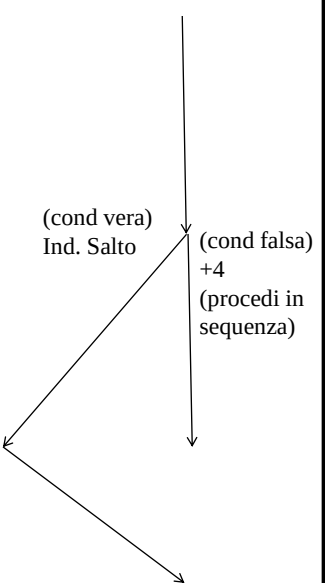
Nome campo	op	rs	rt	costante
Dimensione	6-bit	5-bit	5-bit	16-bit
`addi $s1, $s2, 64`	001000	10010	10001	0000 0000 0000 0100

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
```


Istruzioni di salto condizionato

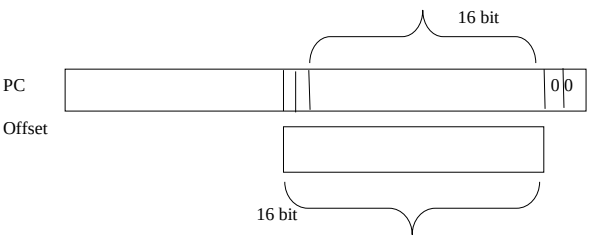
- Salti condizionati relativi:
  - **`beq r1, r2, L1`** (*branch on equal*)
  - **`bne r1, r2, L1`** (*branch on not equal*)
- Salti condizionati relativi:
  - Il flusso sequenziale di controllo cambia solo se la condizione è vera (`beq`)
  - Il calcolo del valore dell'etichetta **`L1`** (**indirizzo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



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 sono multipli di 4.
Calcolo lo spiazzamento in numero di parole invece che di Byte.
Considero 64Kword (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 rappresenta l'offset in termini di numero di istruzioni

Calcolo dell'indirizzo di salto

```
                          0x400          addi $t1, $zero, 10   # N=10
for (i=0; i++; i<10)      0x404          addi $t0,$zero,0      # i =0;
{    ....                 0x408    loop:  addi $t0, $t0,1       # i++
                          0x40C          ....

                          0x420          bne $t0, $t1, loop
}
```

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

Calcolo dell'indirizzo di salto

```
0x400              addi $t1, $zero, 10   # N=10
0x404              addi $t0,$zero,0      # i =0;
0x408    loop:     addi $t0, $t0,1       # i++
0x40C              ....

0x420              bne $t0, $t1, loop
```

L'indirizzo di destinazione è 0x408 (indirizzo dell'etichetta loop)

Lo spiazzamento del salto in byte è pari a: (0x408 - 0x424) = -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 = 0x408 = 0x424 + -7 * 4

5	8	9	-7 = 1111 1111 1111 1001

Istruzioni di tipo I - Branch

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: (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, L1`	000100	10001	10010	1111 1111 1110 0111

L1 = PC+4 -100 byte Codifica su 18 bit: (11)111 1111 1110 0111(00) in binario.

Osservazione

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	indirizzo
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)

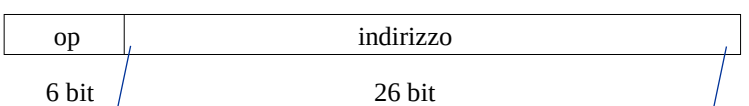
Non c'è corrispondenza 1 a 1, tra tipi strutturali e tipi funzionali.

Il linguaggio macchina

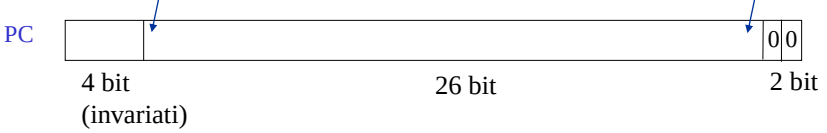
- I diversi tipi di istruzioni: Istruzioni di tipo I
- I diversi tipi di istruzioni: Istruzioni di tipo J
- Register file

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 = 256 Mbyte (segmento testo).

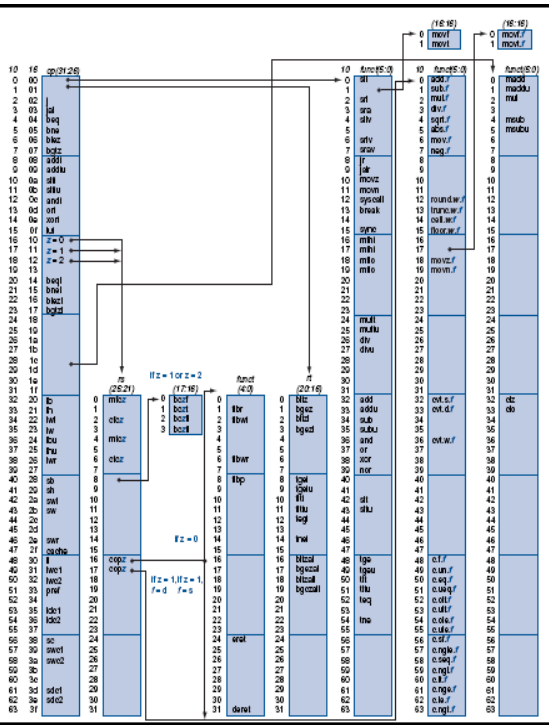


$0 \leq \text{indirizzo} < 2^{28} = 256 \text{ MByte}$

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				



La mappa dei codici operativi

FIGURE A.10.2

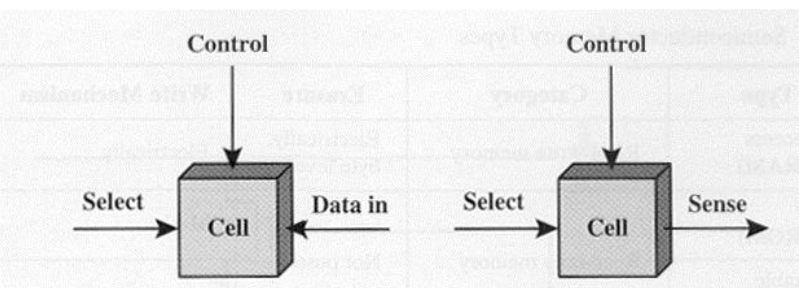
del Patterson Hennessy

Il linguaggio macchina

- I diversi tipi di istruzioni: Istruzioni di tipo I
- I diversi tipi di istruzioni: Istruzioni di tipo J
- Register file

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.

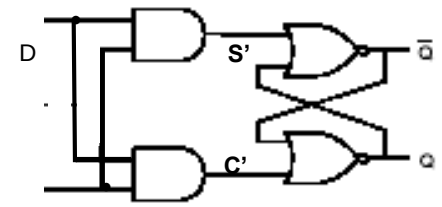


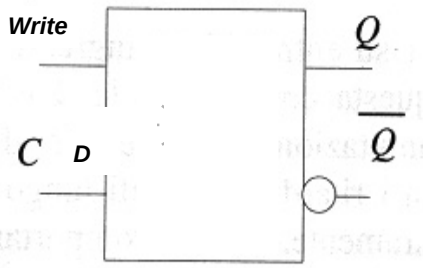
Control (lettura – scrittura)
Select (selezione)
Data in & Sense (Data in & Data out).

Latch sincrono come elemento di memoria

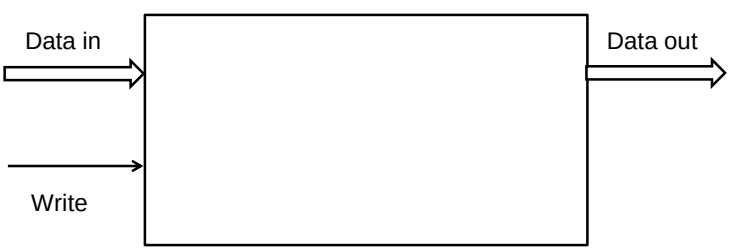
L'ingresso "clock" del bistabile viene utilizzato come segnale di write

E' trasparente quando Write = 1
Se Write = 1 $Q_{t+1} = D$
Se Write = 0 $Q_{t+1} = Q_t$

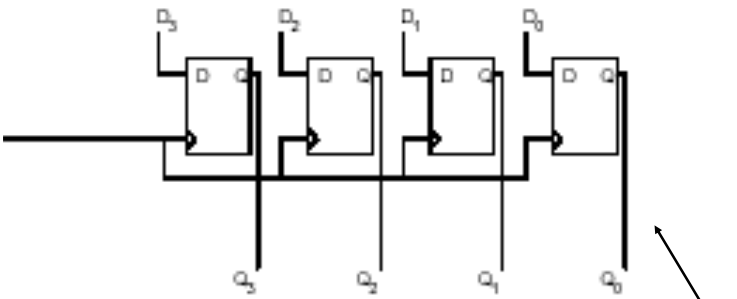




Registro



Registri

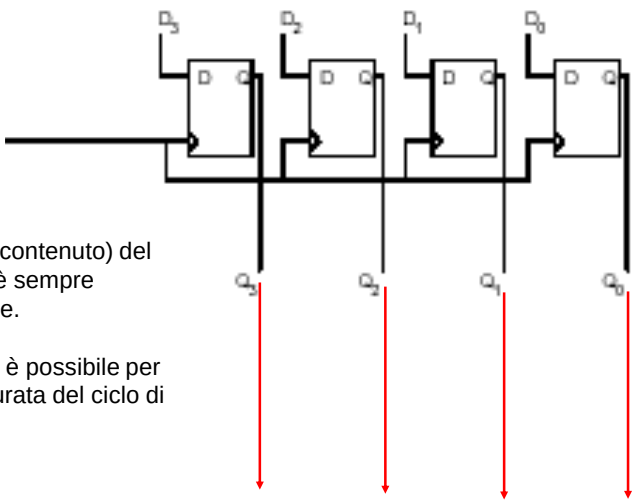


Un registro a 4 bit.
Memorizza 4 bit.

Latch di tipo D

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

Lettura di un registro

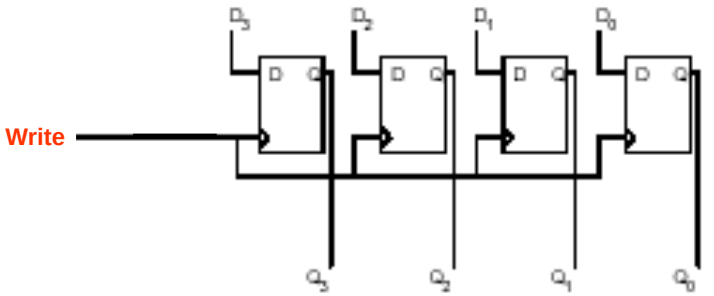


Lo stato (contenuto) del bistabile è sempre disponibile.

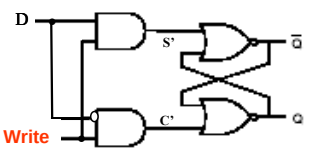
La lettura è possibile per tutta la durata del ciclo di clock.

Scrittura di un registro

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

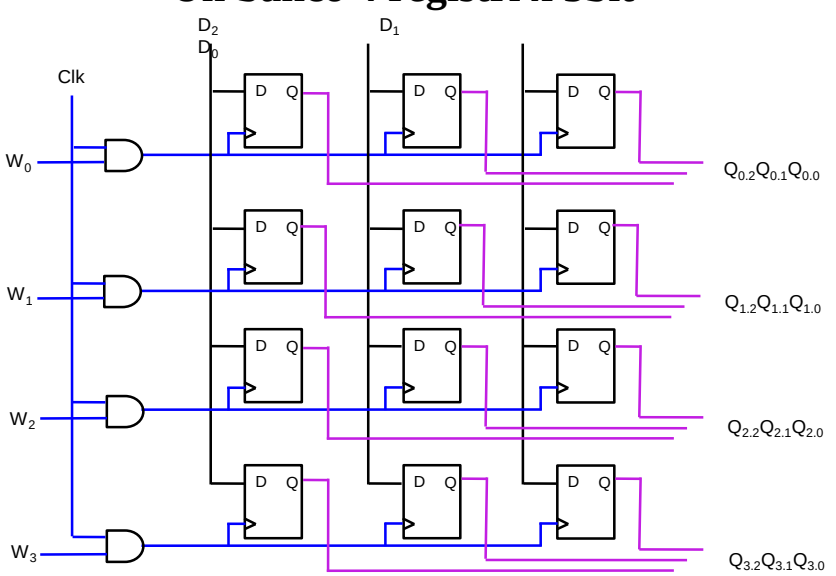


Cosa occorre modificare perchè il registro venga scritto quando serve?
Introdurre una sorta di *"apertura del cancello (chiusura circuito)"* .
Può essere sincronizzata o meno con il clock.

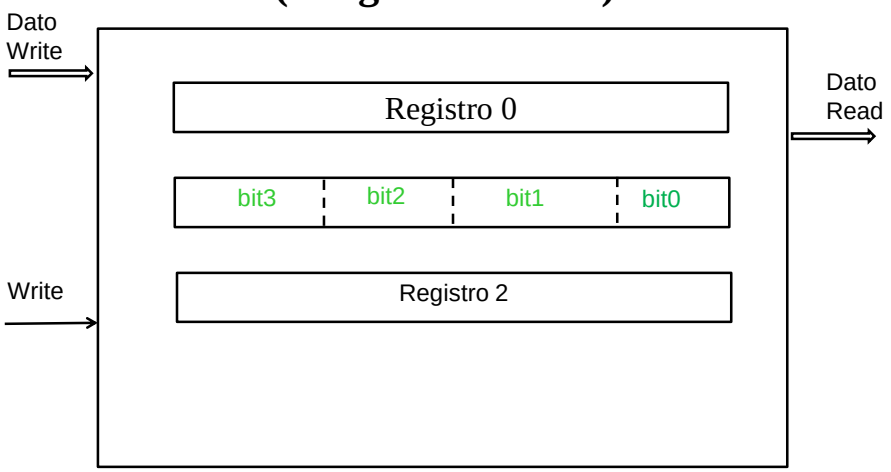


Il clock apre il passaggio al conteuto di D attraverso il latch. Quando il segnale di Write è a zero, lo stato non varia.

Un banco 4 registri x 3bit

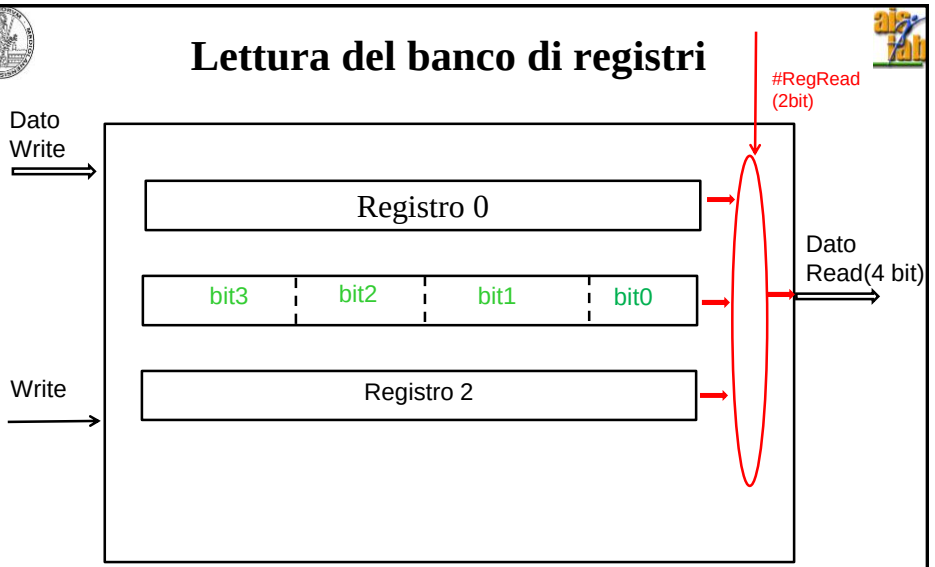


Funzionamento di un banco di registri (3 registri da 4 bit)



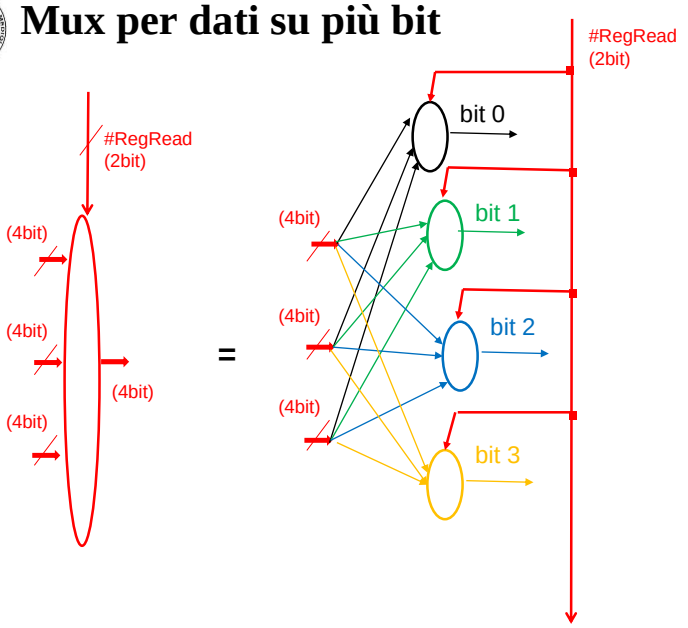
Come seleziono il dato da leggere fra i 3 dati possibili?
Come seleziono il registro su cui scrivere fra i 3 registri?

Lettura del banco di registri

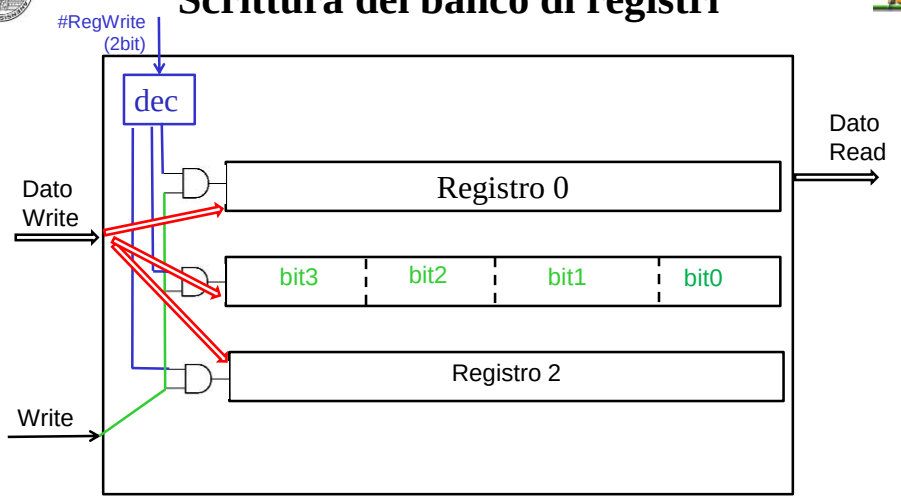


Seleziono uno dei registri = porto in uscita l'uscita Q di tutti i bit del registro selezionato
Avrò tanti Mux quanti sono i bit che costituiscono il registro (in questo caso .....)

Mux per dati su più bit



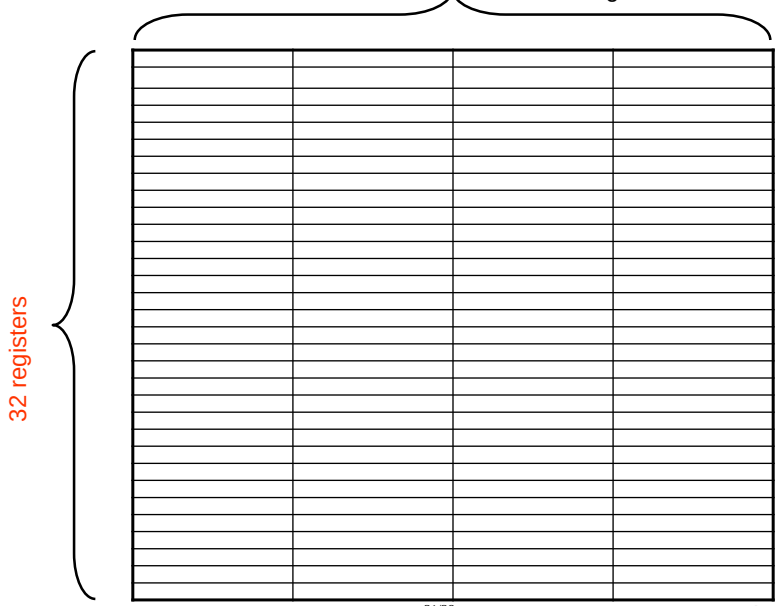
Scrittura del banco di registri



Invio il dato da scrivere a tutti i registri ma devo inviare il segnale di scrittura solamente al registro selezionato

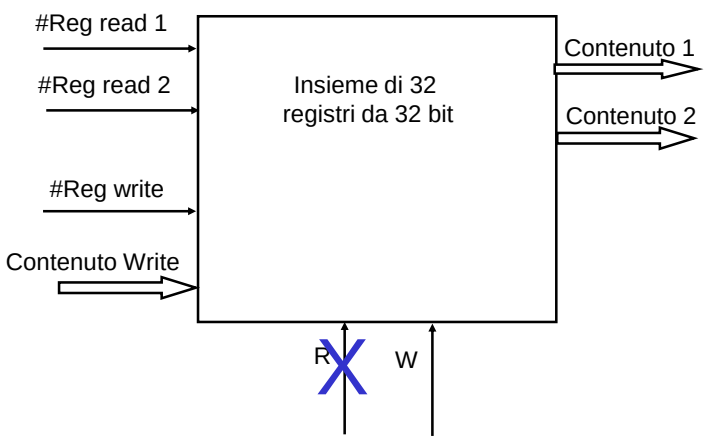
Banco di Registri

Struttura costituita da un insieme di registri



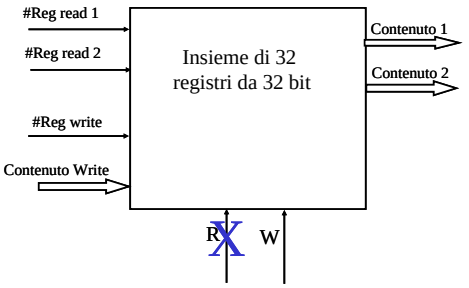
Register file

Banco di registri utilizzabile come memoria



Possono essere letti / scritti fornendo il numero del registro.

Gestione del register file

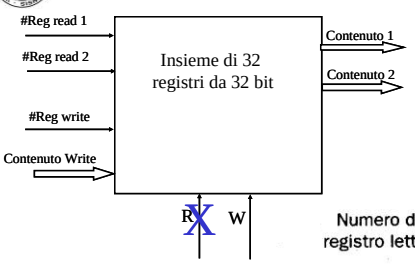


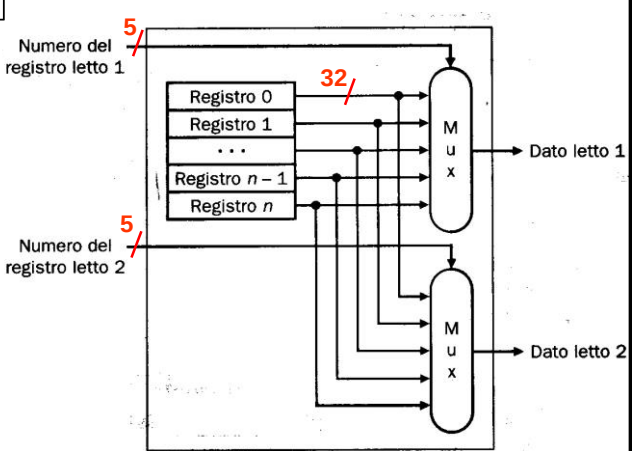
La lettura non modifica il contenuto di un registro (collego uscita Slave con il circuito combinatorio).

La scrittura invece richiede la modifica. Occorre il segnale W.

#bit_indirizzamento = $\log_2$ #bit

Porta di lettura del register file

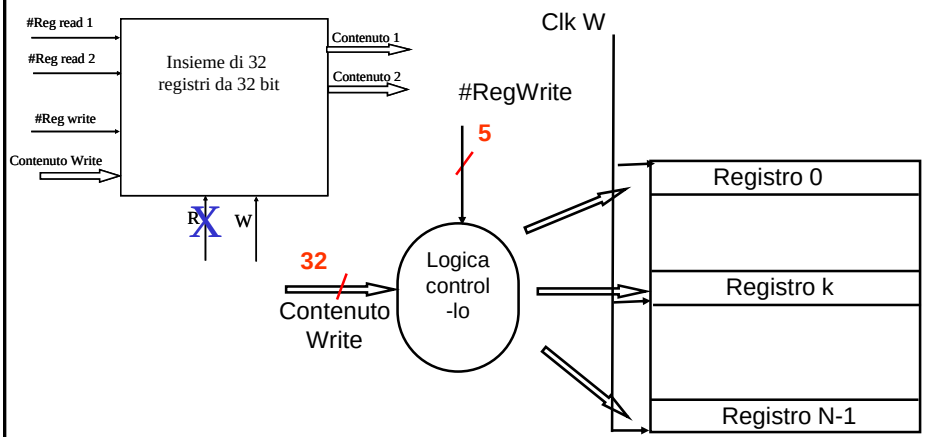




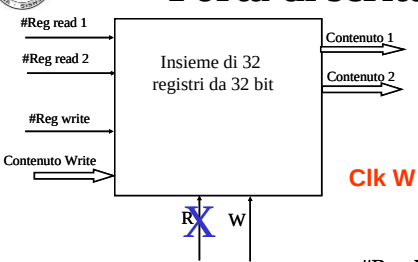
Un mux per ogni porta di lettura.

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

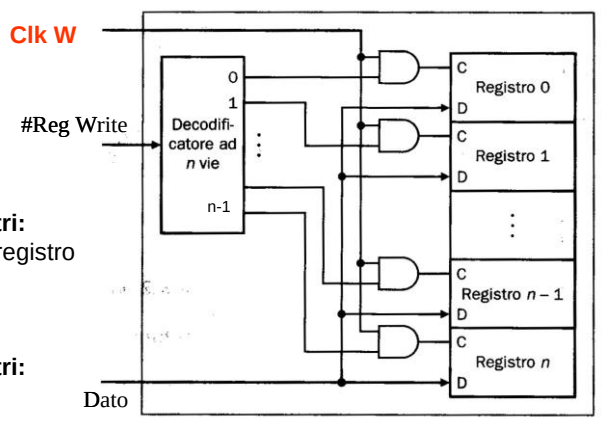
Possibile porta di scrittura del register file



Porta di scrittura del register file



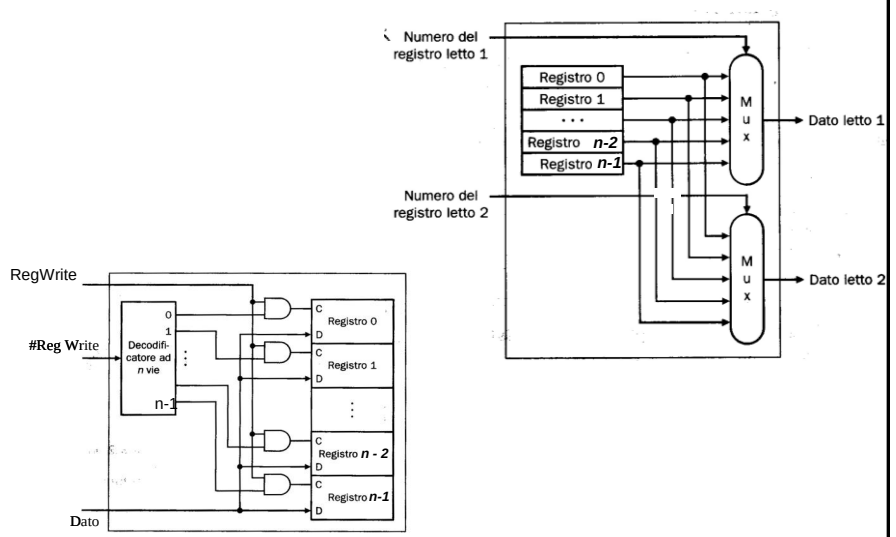
Ho ridotto drasticamente il numero di porte AND.



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

Ingresso D del latch dei registri:
Bit dato corrispondente.

Register file



Il linguaggio macchina

- I diversi tipi di istruzioni: Istruzioni di tipo I
- I diversi tipi di istruzioni: Istruzioni di tipo J
- Register file