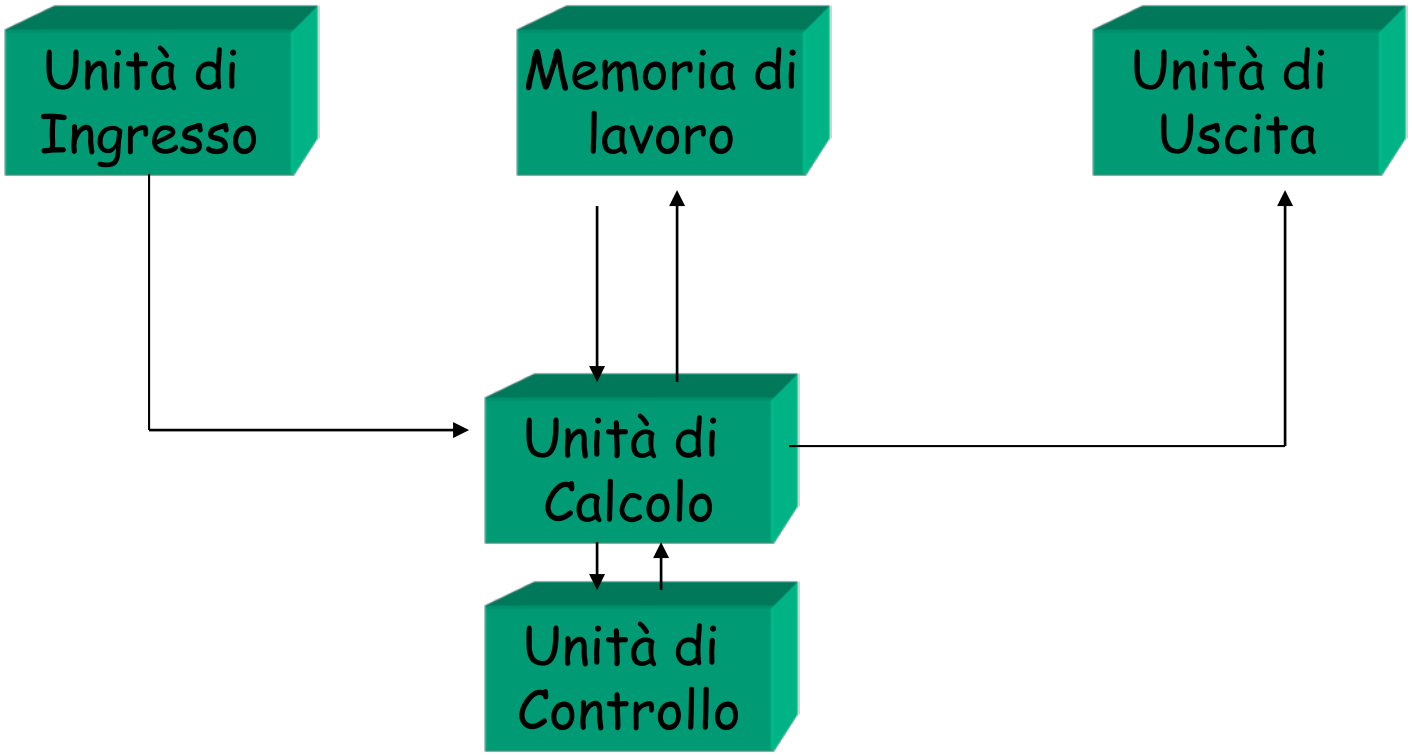
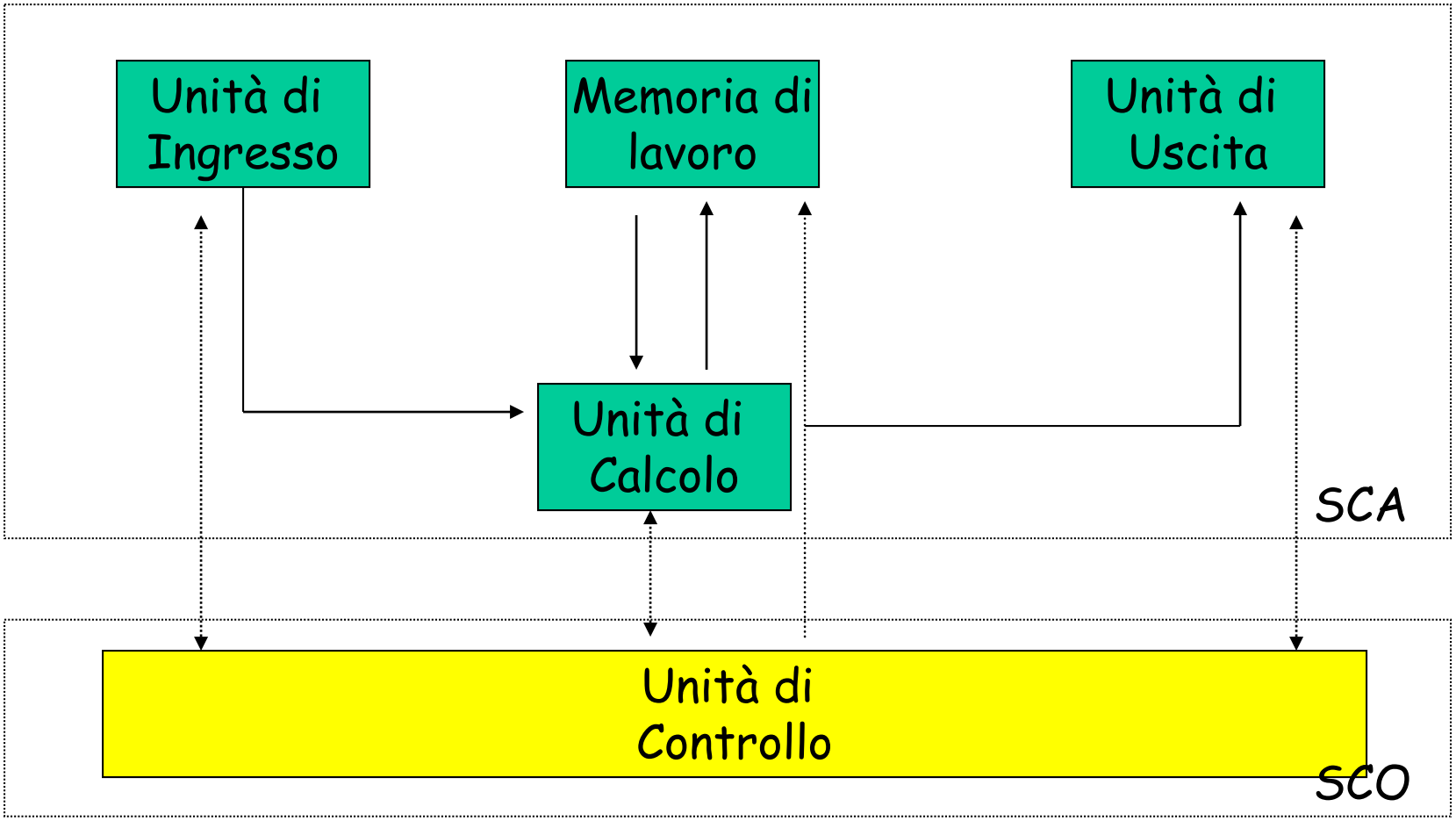
Il processore z64

Macchina di von Neumann



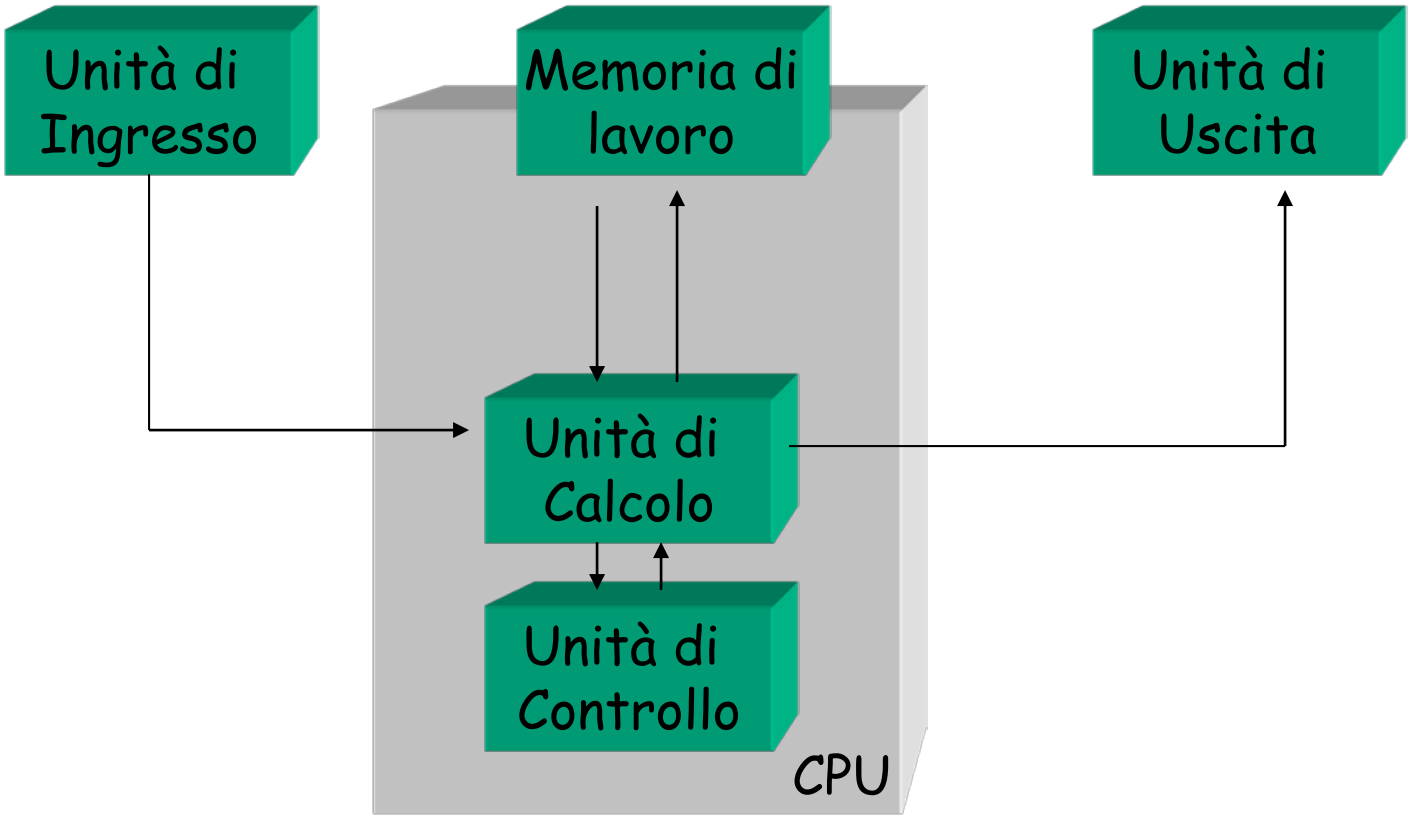
Suddivisione SCA-SCO



Segnali di controllo/condizione

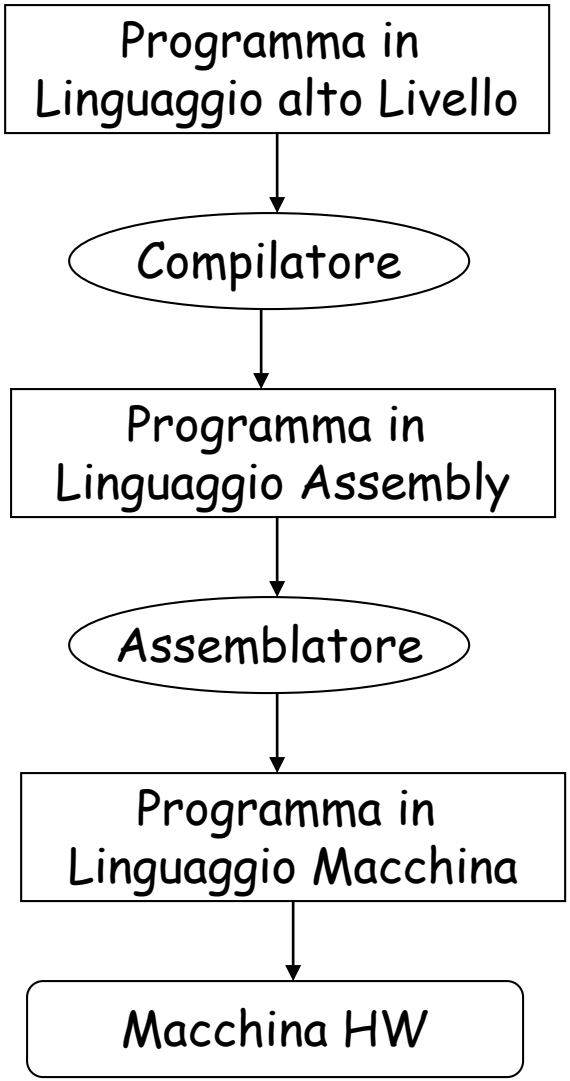
Flusso dati

Modifica macchina di Von Neumann con CPU



Dal linguaggio ad alto livello al linguaggio macchina

Non dipende dalla macchina HW



a=b+c

- Insieme istruzioni che dipendono dalla macchina hw (simboliche)
- Commenti
- Riferimenti simbolici

movw b,R1
movw c,R2
addw R2,R1
movw R1,a

- Insieme Istruzioni della macchina hw
- Riferimenti indirizzi fisici

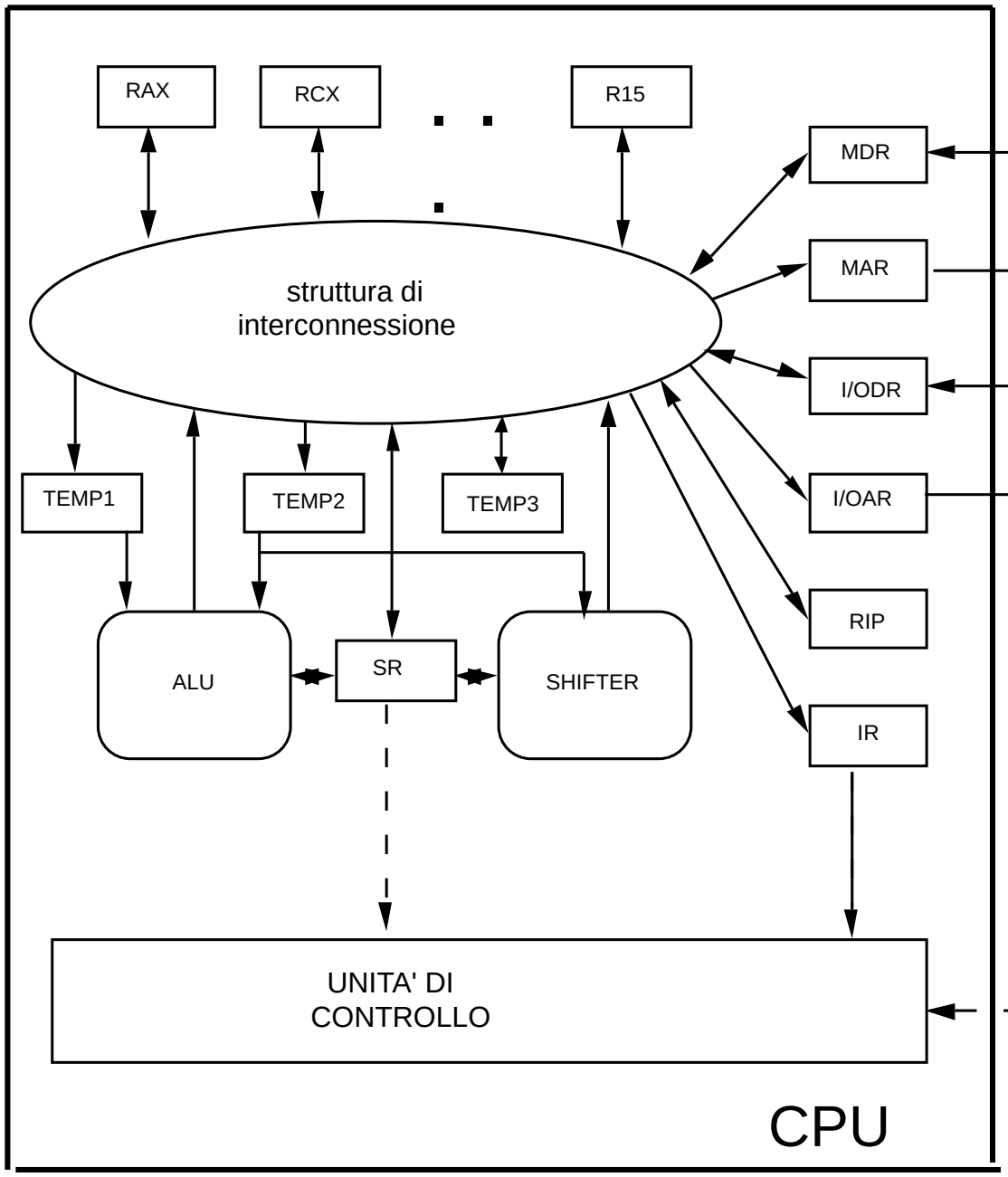
000101..010100
1011101..010100
01011..11101010
010..1110101010

z64

- Processore "virtuale" dotato di registri da 64bit
- Non esiste nella realtà, ma le sue funzionalità sono simulate tramite un programma
- Nel seguito sarà usato per approfondire alcuni aspetti legati alle architetture dei calcolatori
- Sistema multiciclo
- Prima versione:
  - no pipeline,
  - no memoria cache

R0,R1...
registri

ALU,
shifter
unità di
calcolo



Collega-
menti
con
memoria
e I/O

N.B.
non sono indicati i segnali di controllo che dal SCO vanno verso i registri interni, le unità di calcolo e la struttura di interconnessione

z64- Sottosistema di Calcolo (SCA)

- Registri (basati su Flip-Flop D con segnale di *Enable*)
  - speciali
  - generali
- Dispositivi di calcolo
  - Shifter
  - ALU (somma e sottrazione)
- MUX
- Decodificatori
- Struttura di interconnessione: BUS

Esecuzione dell'istruzione

L'esecuzione dell'istruzione prevede tre fasi:

- Il ***fetch*** serve a prelevare l'istruzione dalla memoria
- La ***decodifica*** è fatta dal sistema di controllo
- L'***esecuzione*** di una istruzione consiste in un numero variabile di operazioni elementari che possono essere:
  - Accesso ai dati memorizzati nei registri interni
  - Calcolo dell'indirizzo di memoria da cui leggere e/o scrivere i dati
  - Accesso in lettura in memoria
  - Manipolazione dei dati
  - Accesso in scrittura in memoria
  - Lettura dei dati da periferica
  - etc

Ciclo istruzione

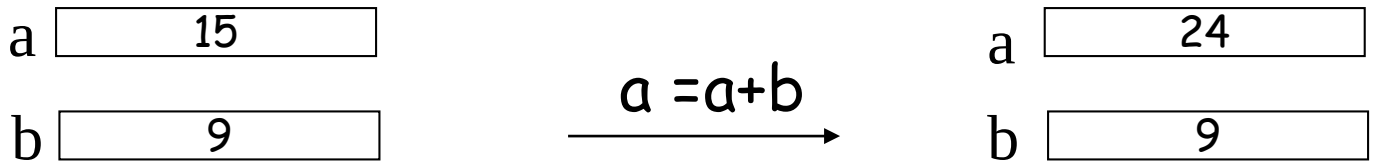
Le interazioni elementari del processore con il mondo esterno (memoria o dispositivi di input/output) vengono denominate ***CICLI MACCHINA***

Ogni ciclo istruzione prevede almeno un ciclo macchina (il fetch)

Un semplice esempio

Consideriamo l'istruzione a=a+b, espressa in un linguaggio di alto livello

- "Memorizza nella variabile di nome a, la somma dei valori contenuti nelle variabili di nome a e b"
  - Nota: Le variabili sono individuate da un nome simbolico deciso precedentemente nel programma..



Prima Dopo

Un semplice esempio (2)

Per eseguire questa istruzione è necessario

- Stabilire dove sono memorizzati i valori da sommare
- Stabilire dove va scritto il risultato dell'operazione
- Quale operazione svolgere

- Nello z64, gli operandi sono memorizzati nei registri interni alla CPU (registri visibili al programmatore)
- Il formato dell'istruzione è (*s* può essere B,W,L,Q)

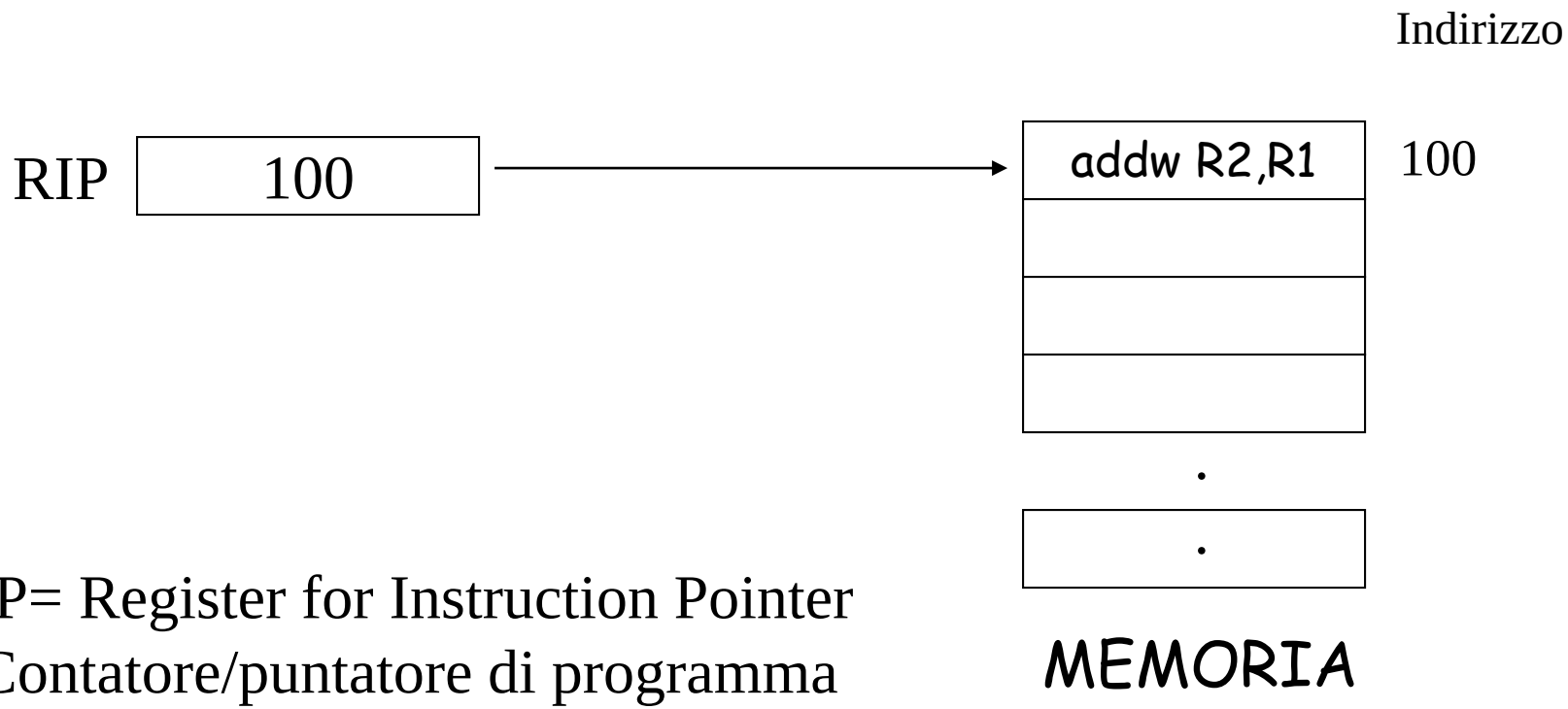
ADD*s* <sorgente>,<destinazione>

Per esempio

ADDW R2,R1

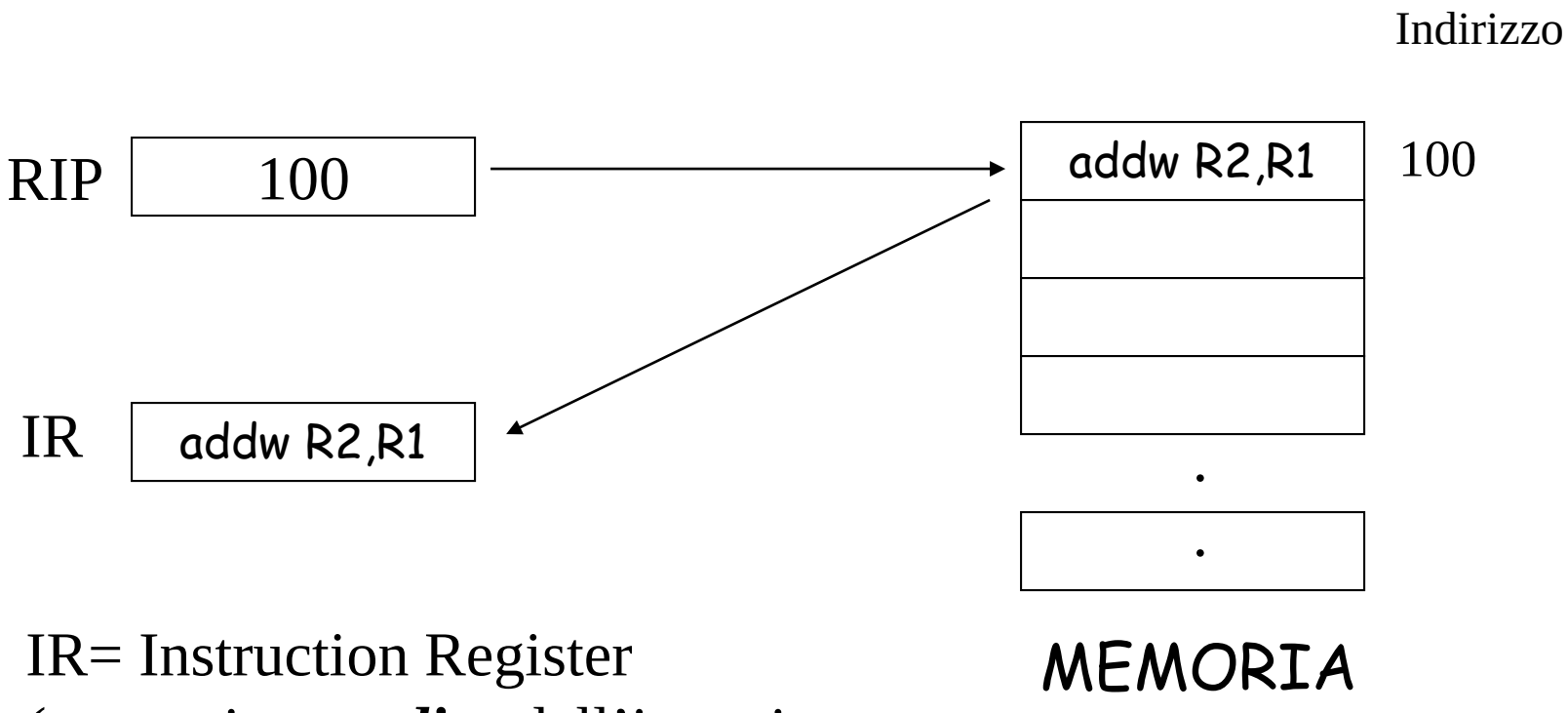
#somma il contenuto di R1 con quello di R2 (32 bit) e poni il risultato in R1

Esecuzione istruzione



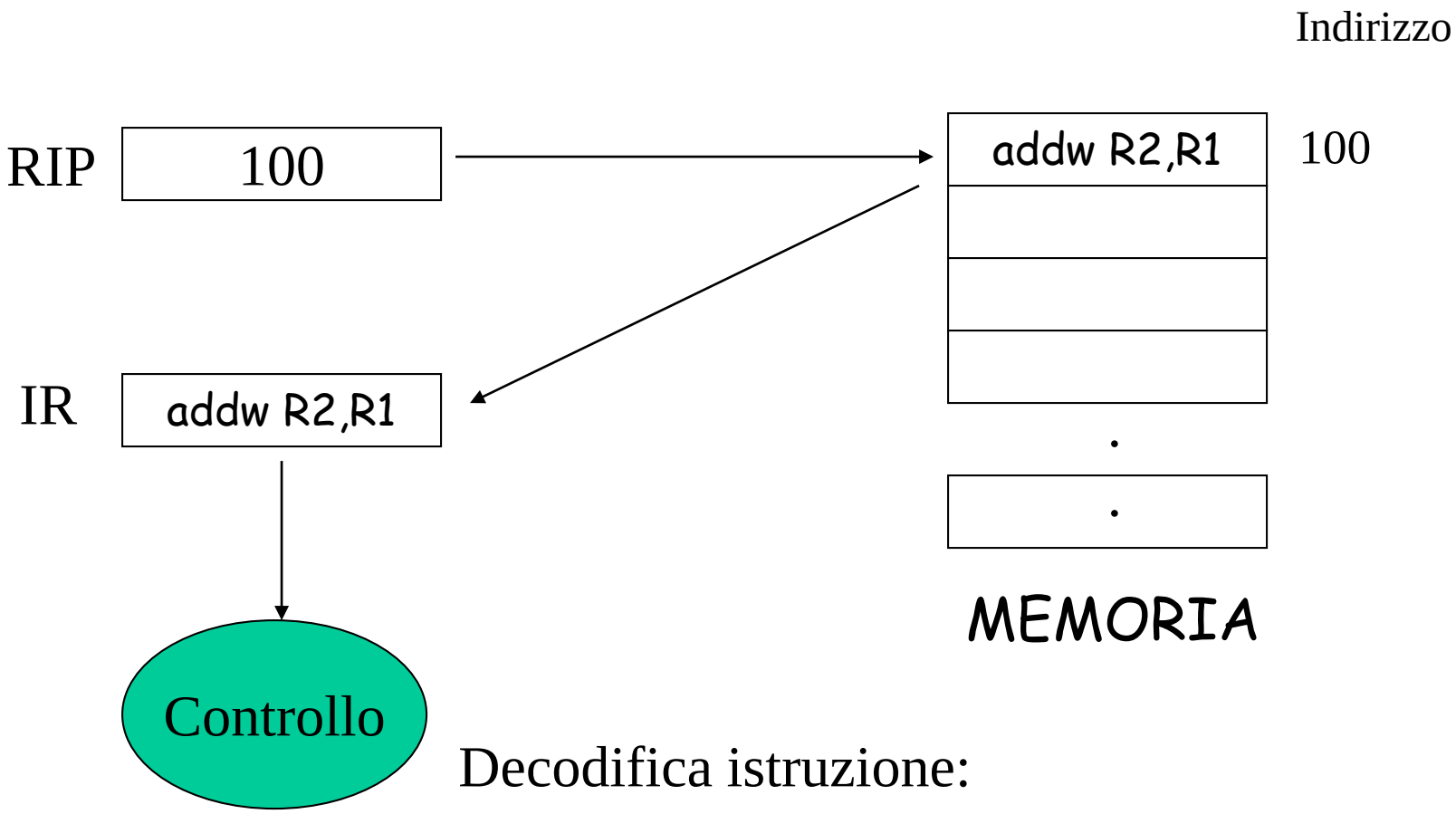
RIP= Register for Instruction Pointer
Contatore/puntatore di programma
(memorizza indirizzo dell' istruzione
in esecuzione)

Istruzione: fase di fetch



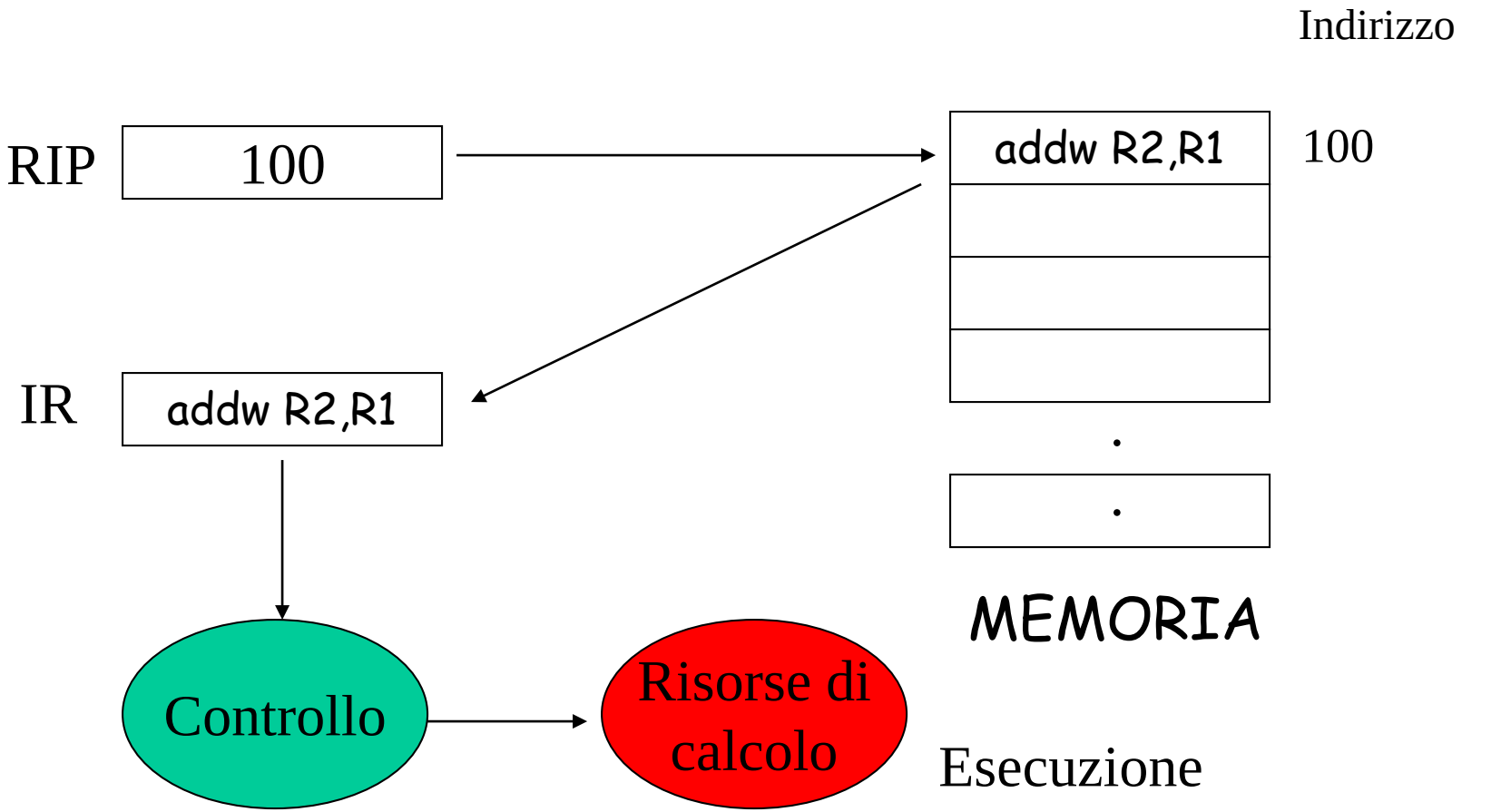
IR= Instruction Register
(memorizza ***codice*** dell'istruzione
in esecuzione)

Istruzione: decodifica



Decodifica istruzione:
predispone i comandi
per la sua esecuzione

Istruzione: fase di esecuzione



Esecuzione dell'istruzione (consiste di diversi passi)

CPU come interprete

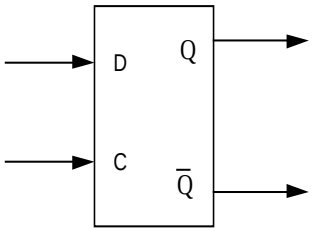
- La CPU interpreta le istruzioni che man mano sono presenti nel suo Instruction Register (notare che l'esecuzione di una istruzione di salto puo' modificare il contenuto di RIP)

fetch: (RIP)→IR
incrementa RIP
esegui istruzione in IR
vai al passo *fetch*

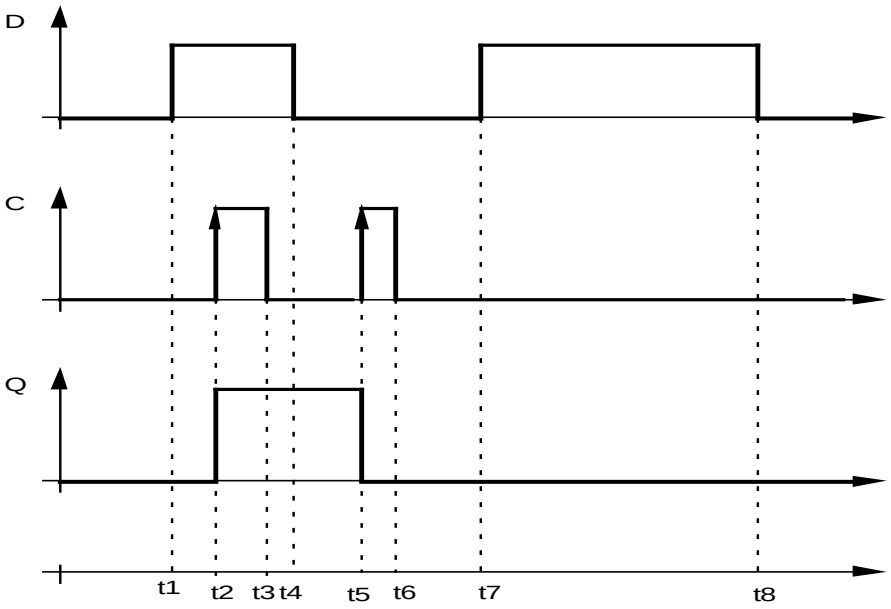
La notazione (RIP) indica il contenuto della locazione di memoria con indirizzo RIP

- Tale schema è semplificato poiché per interagire con l'esterno, o gestire situazioni anomale, tale ciclo deve poter essere interrotto.

Registri interni (realizzati con flip/flop di tipo D)

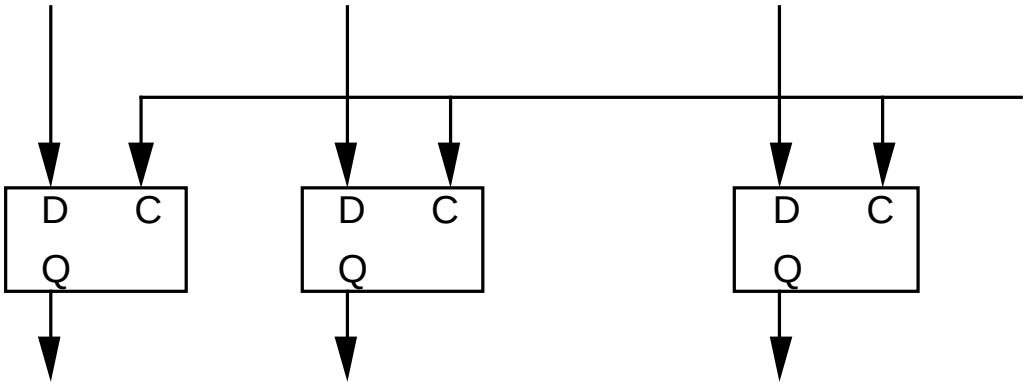


Simbolo grafico del flip-flop di tipo D

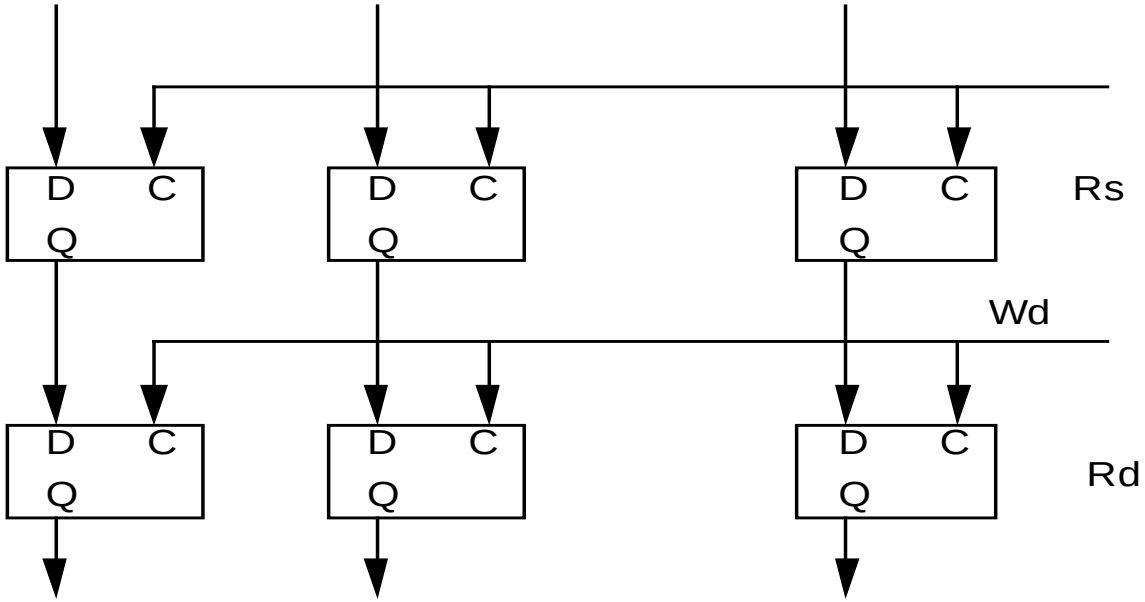


Relazione ingresso/uscita
flip-flop di tipo D commutante sul fronte di salita del segnale di abilitazione

Schema di registro

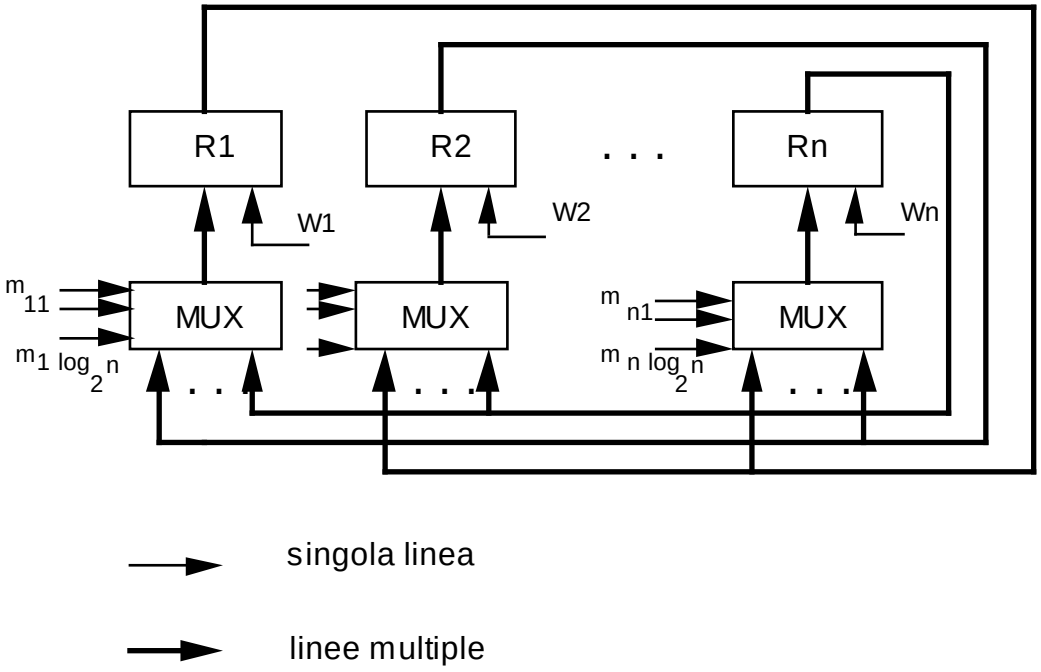


Interconnessione diretta tra registri



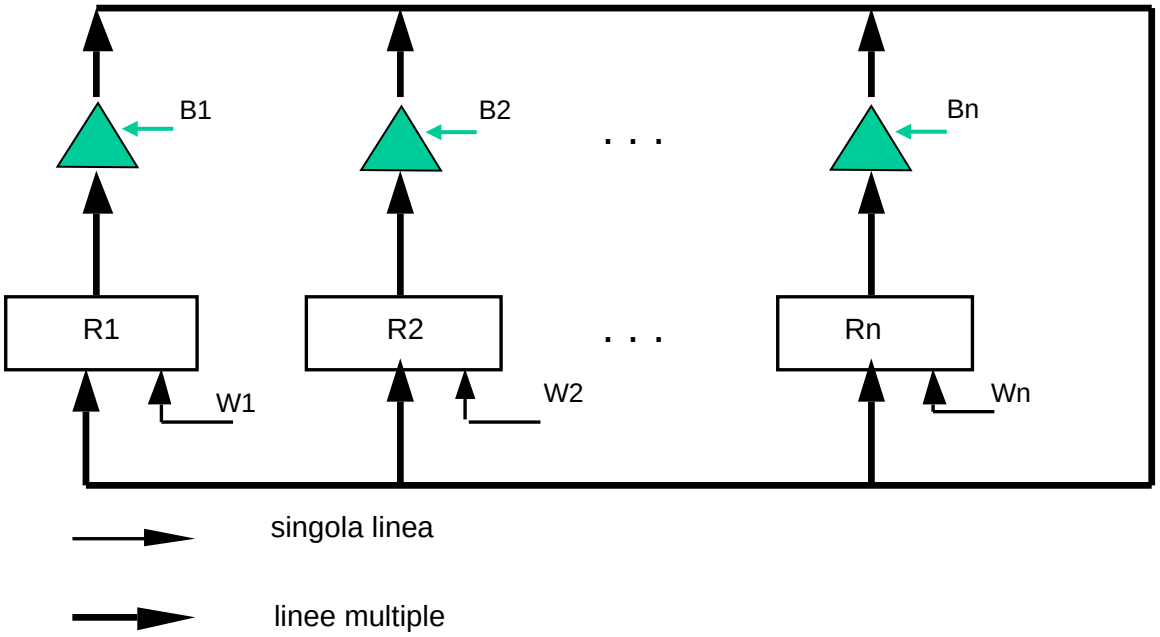
- Vantaggi:
  - semplice da implementare
- Svantaggi:
  - difficoltà di interconnettere più registri

Interconnessione tramite multiplexer



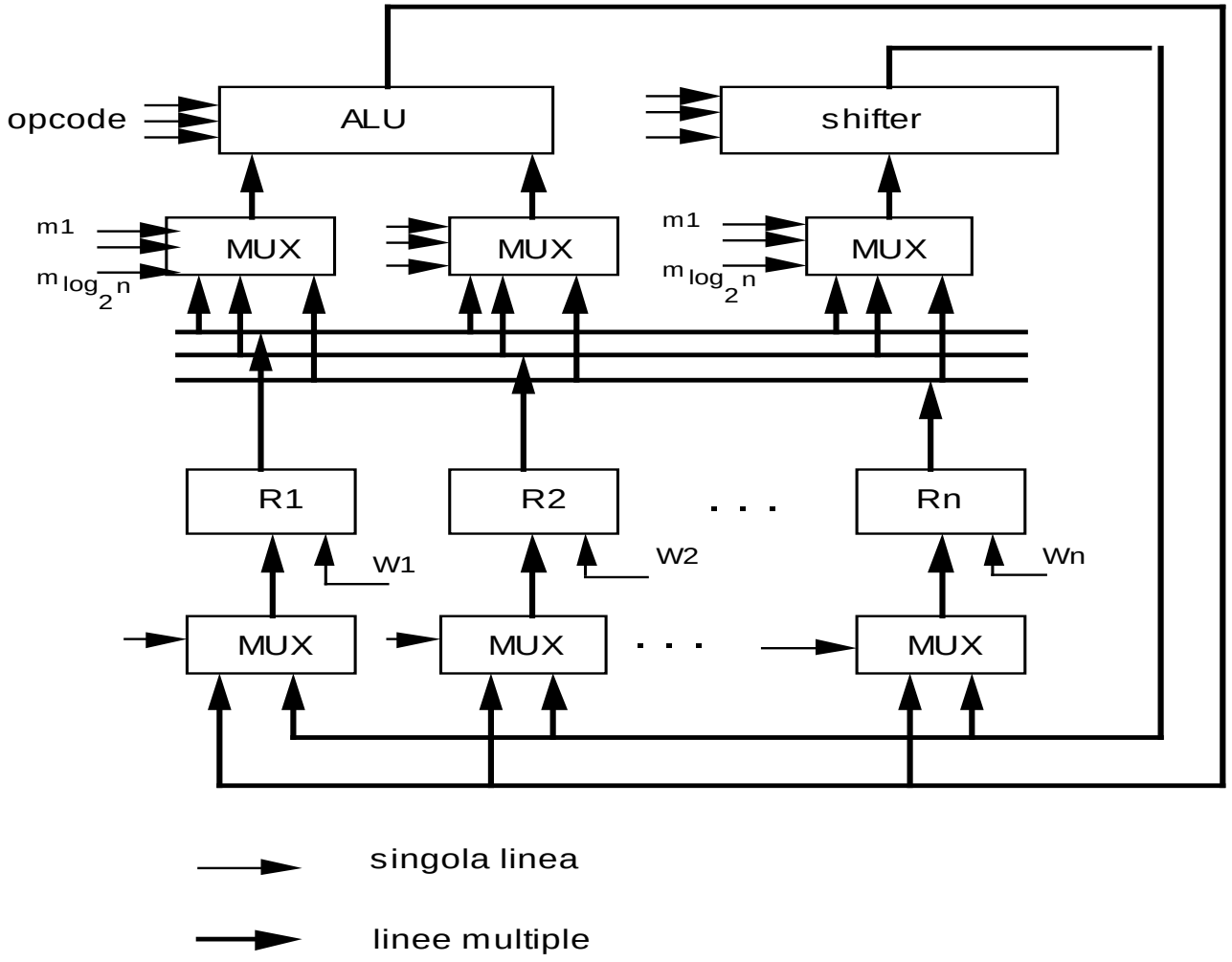
- Vantaggi:
  - semplice da implementare
  - possibilità di trasferire più dati contemporaneamente
- Svantaggi:
  - costo eccessivo dei multiplexer e delle linee di interconnessione

Interconnessione tramite bus

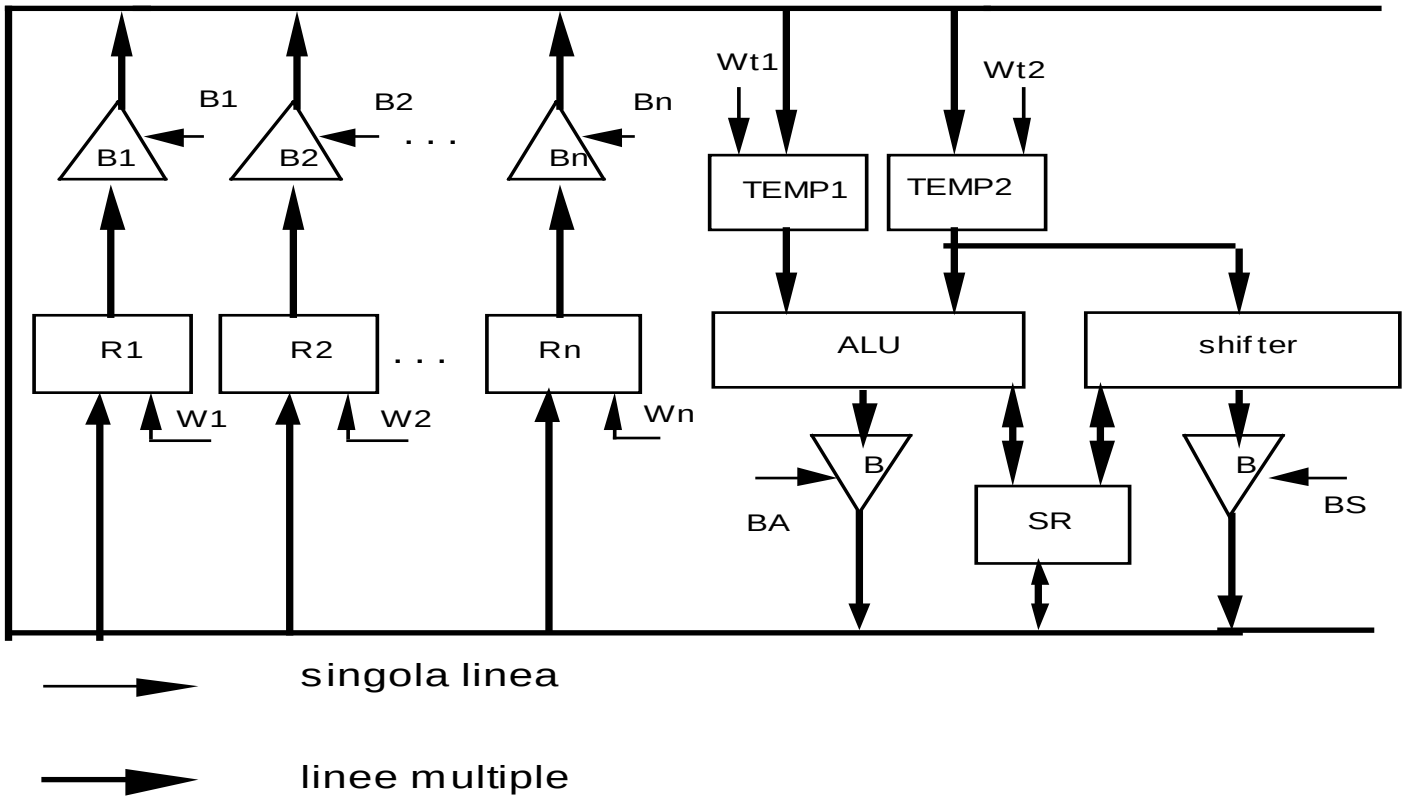


- Vantaggi:
  - semplice da implementare
  - costo contenuto
- Svantaggi:
  - è possibile un solo trasferimento alla volta
  - eccessivo numero di segnali di controllo

Trasferimento dati tra registri e circuiti di calcolo (con multiplexer)



Trasferimento dati tra registri e circuiti di calcolo (con singolo bus)

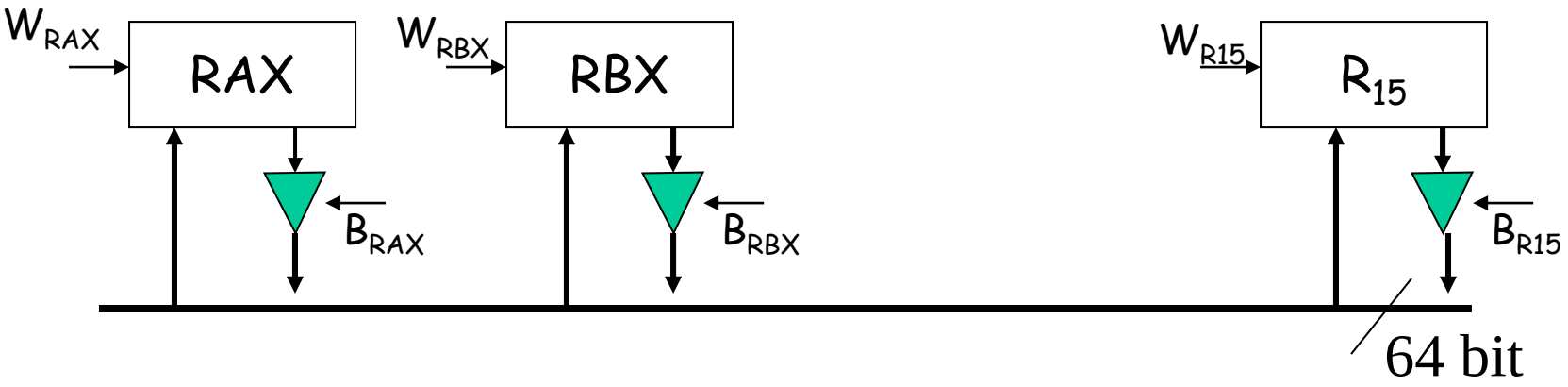


z64: BUS interno

- Usato per il collegamento dei registri interni
- Operazioni che caratterizzano il bus
  - Ricezione dati
    - i bit presenti sul bus sono memorizzati in un registro
  - Trasmissione dati
    - Il contenuto di un registro è posto sul bus
- Al più un solo registro può scrivere sul bus
  - segnali di controllo opportunamente generati
    - Il segnale di abilitazione alla scrittura di un registro corrisponde alla ricezione dei dati presenti sul bus in quel momento
    - Il segnale di abilitazione sul buffer three-state permette di trasferire sul bus il contenuto del registro

z64: BUS interno, segnali di controllo
hp: operandi a 64 bit

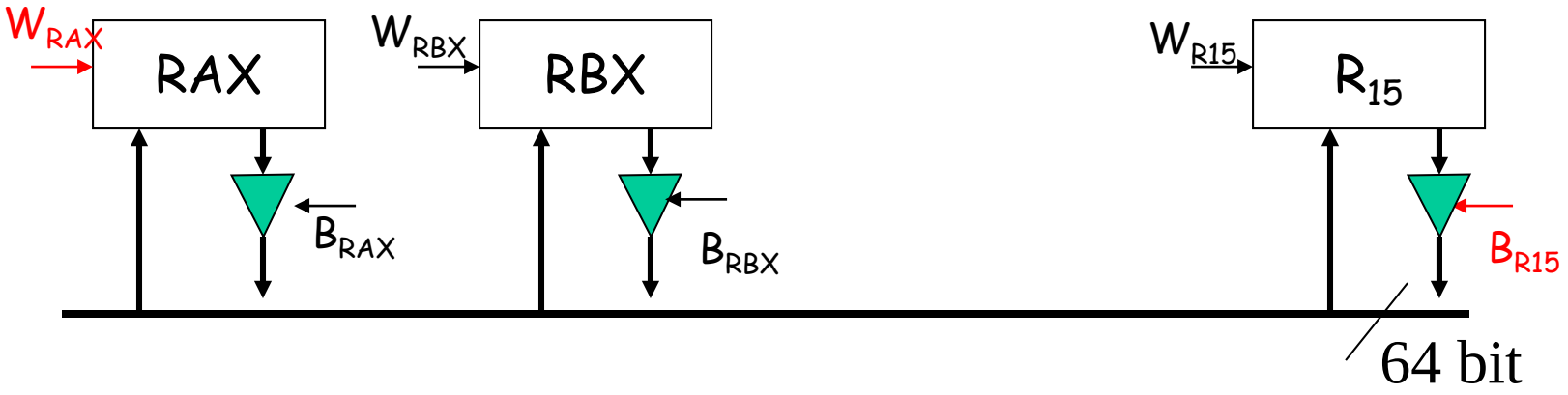
Una sola scrittura per volta (controllo mediante Bi)
2n segnali di controllo (n numero dei registri)



W_i=1, scrivi il contenuto del bus sul registro i
B_i =1 scrivi sul bus il contenuto del registro i

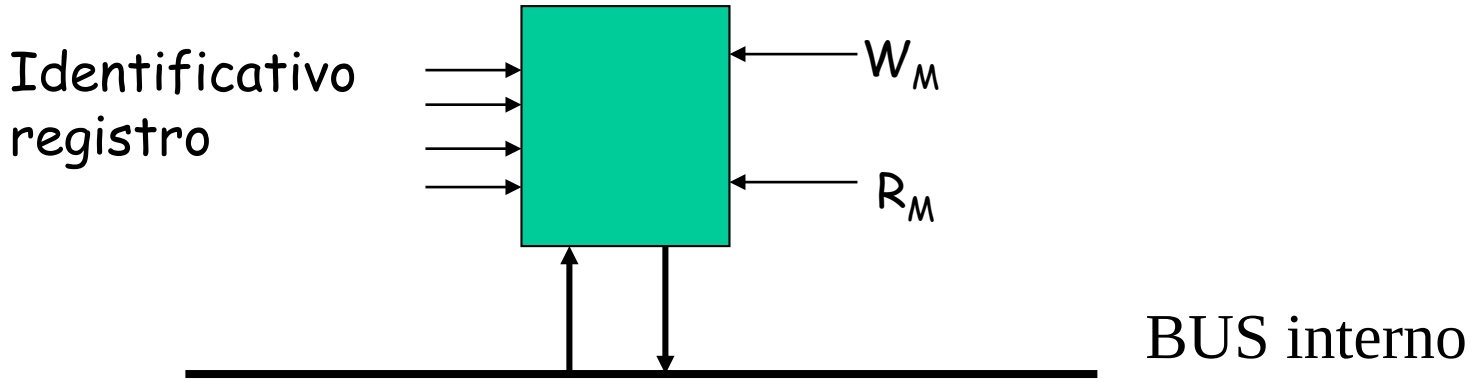
z64: BUS interno, esempio RAX -> R15

Per eseguire il trasferimento da RAX ad R15 (simbolicamente RAX->R15) devono essere affermati solamente i seguenti segnali: $B_{RAX} = 1$, $W_{R15} = 1$

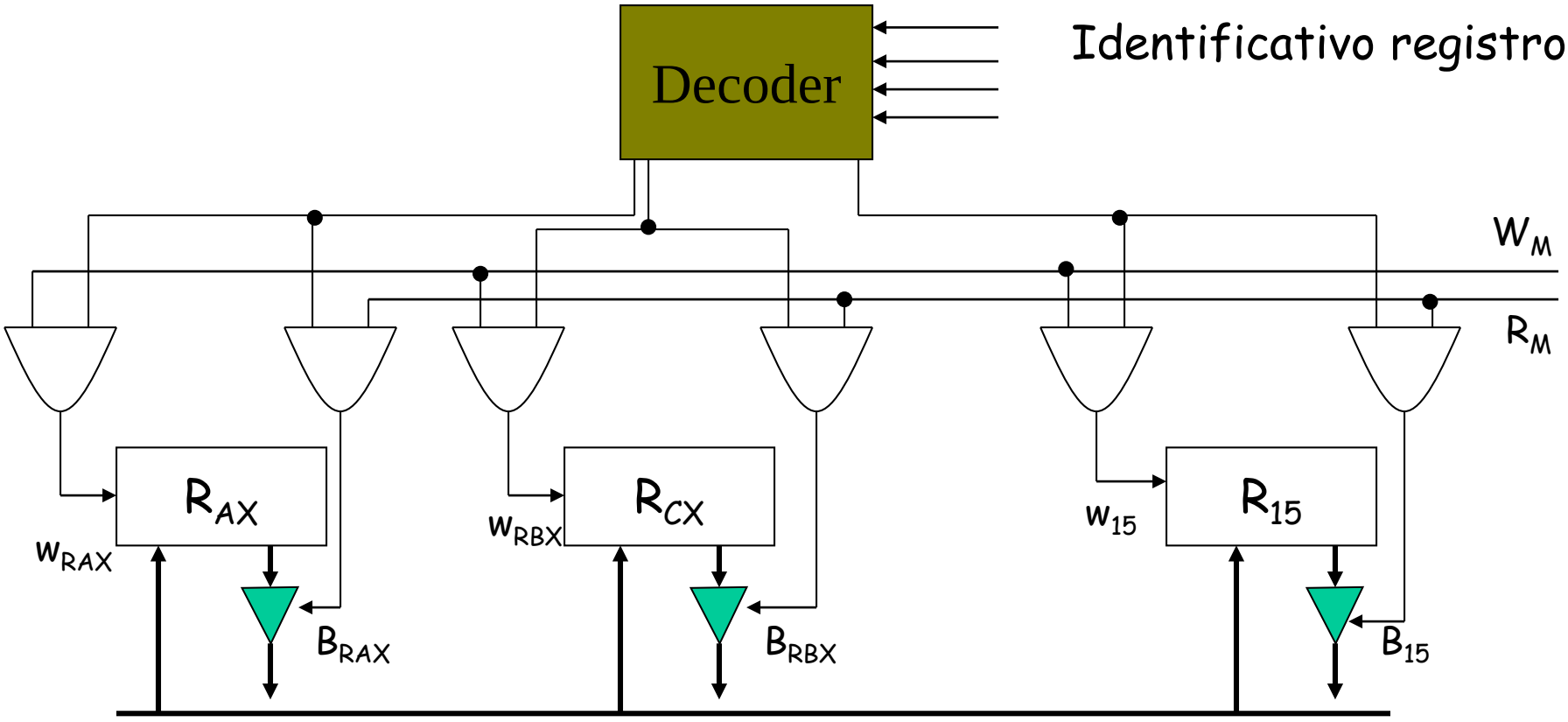


z64 – Banco dei registri

- Insieme di 16 registri generali (RAX, RCX, . . ., R8,. . . R15)
- Sono controllati mediante segnali di abilitazione per
  - scrittura del registro (W_M)
  - lettura e conseguente invio sul bus interno del contenuto del registro (R_M)

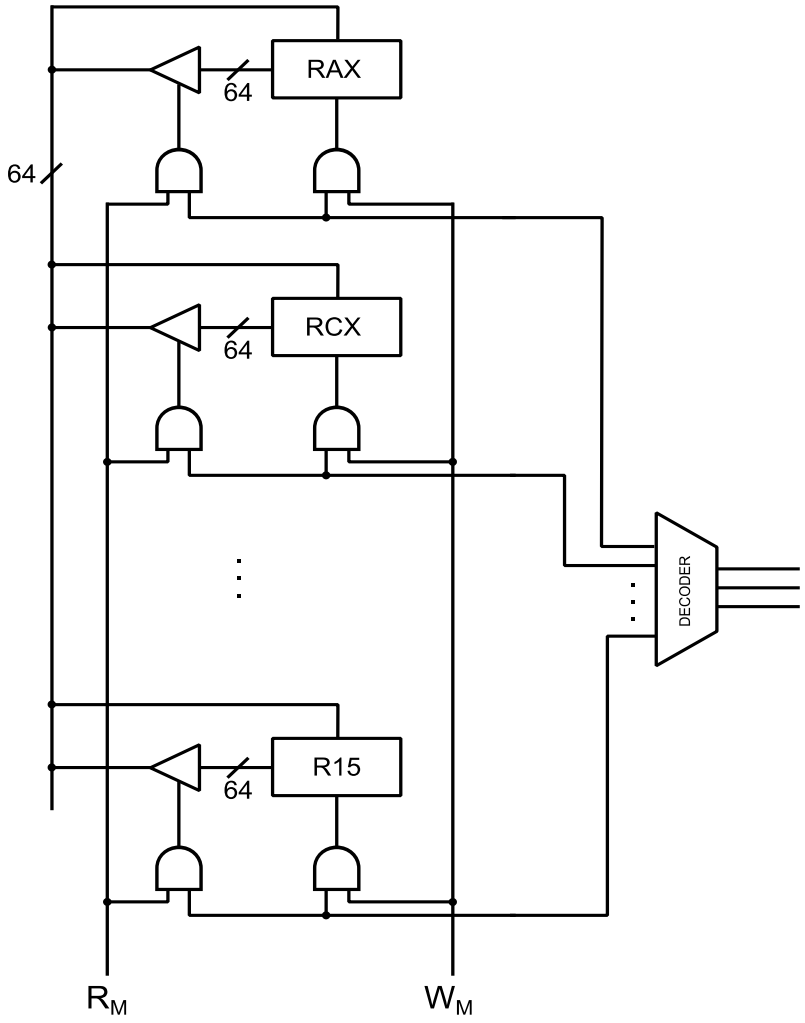


z64: Banco dei registri (a 64 bit)



w_i =1, scrivi valore sul bus in Ri
B_i = 1, invia sul bus valore di Ri

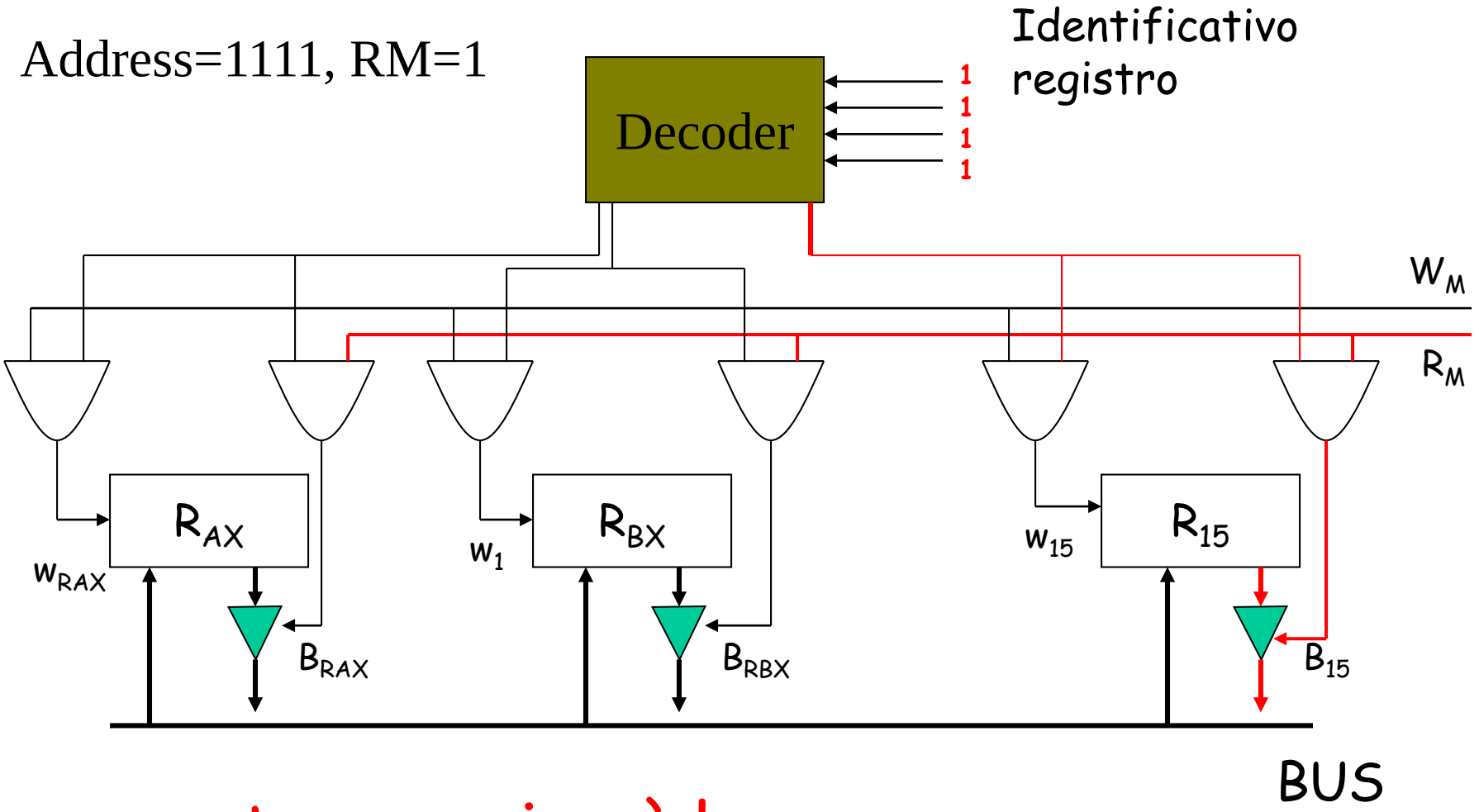
z64: Banco dei registri (a 64 bit)



Codifica identificativi registri

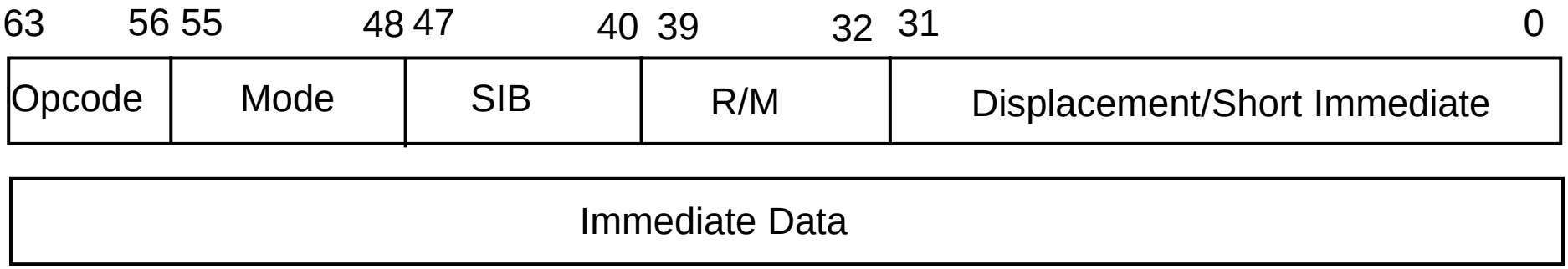
RAX	0000
RCX	0001
RDX	0010
RBX	0011
RSP	0100
RBP	0101
RSI	0110
RDI	0111
R8	1000
R9	1001
R10	1010
R11	1011
R12	1100
R13	1101
R14	1110
R15	1111

z64 - esempio: R15 ->BUS



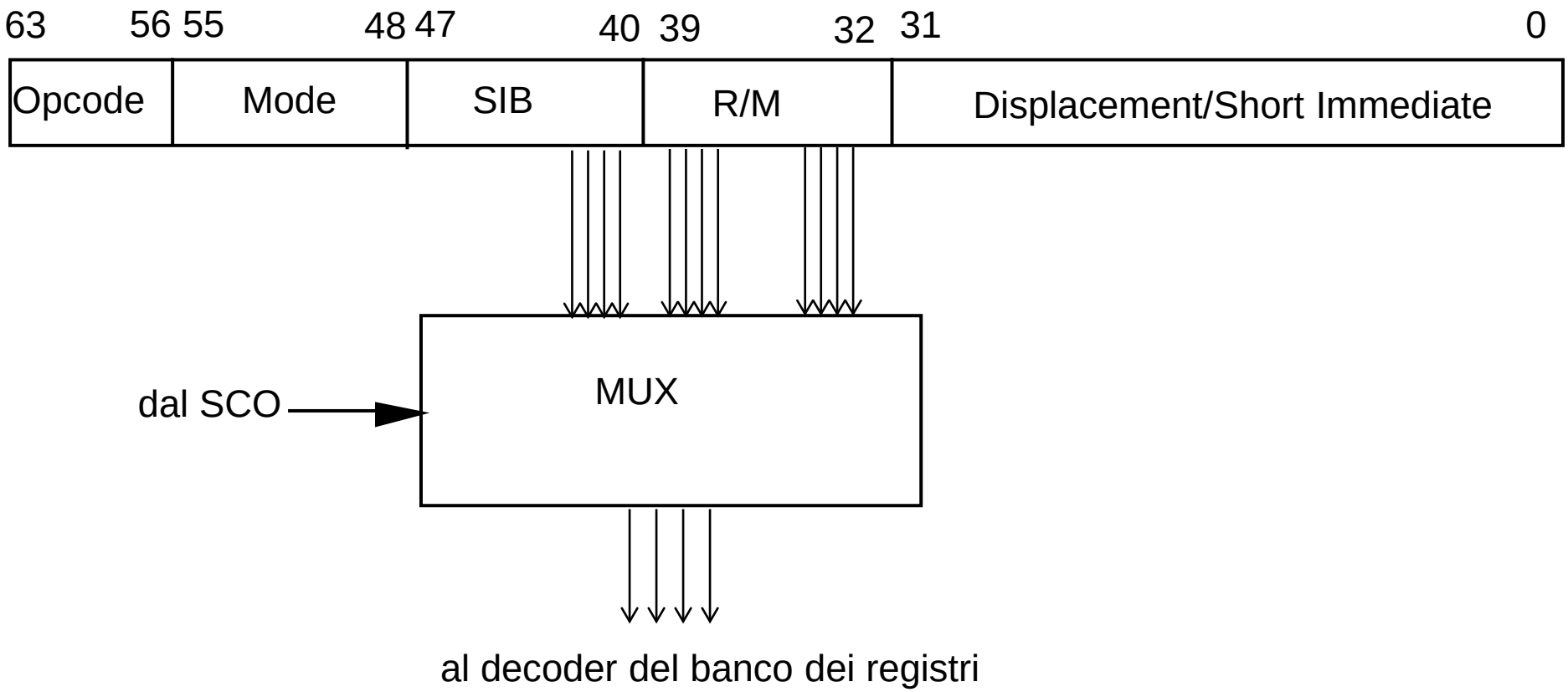
nota: non si può leggere e scrivere contemporaneamente

Formato istruzione a 64/128 bit

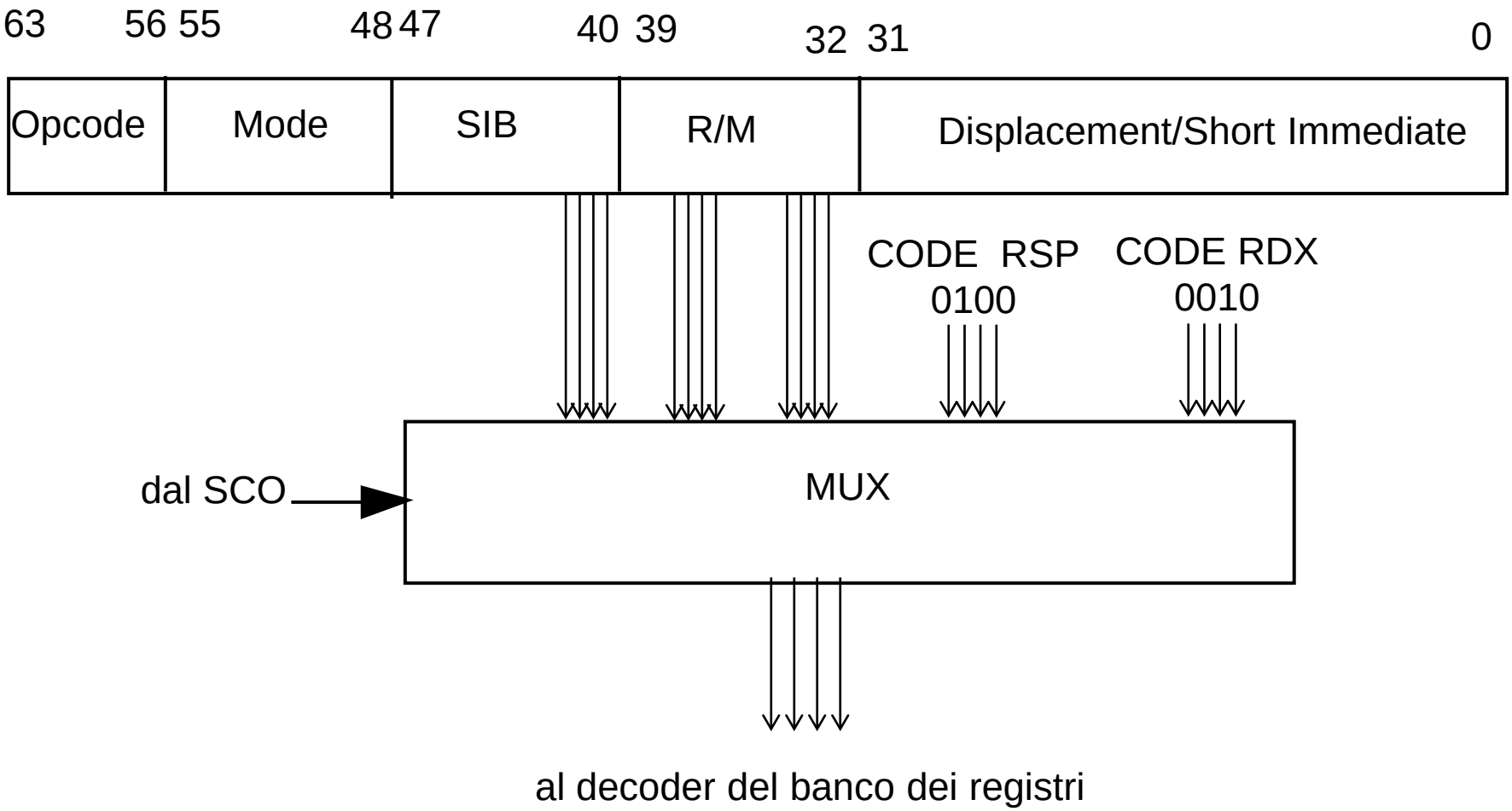


Opcode: codice operativo
Mode: specifica la dimensione degli operandi
SIB (Scale, Index, Base): usato per gli indirizzamenti in memoria
R/M (Register/Memory): specifica dove trovare gli operandi
Displacement/Short Immediate: Indica se c'è un displacement nell'accesso in memoria o il valore dell'immediato a 32 bit
Immediate Data: valore del dato immediato a 64 bit

Switch dell'identificativo dei registri dall'IR

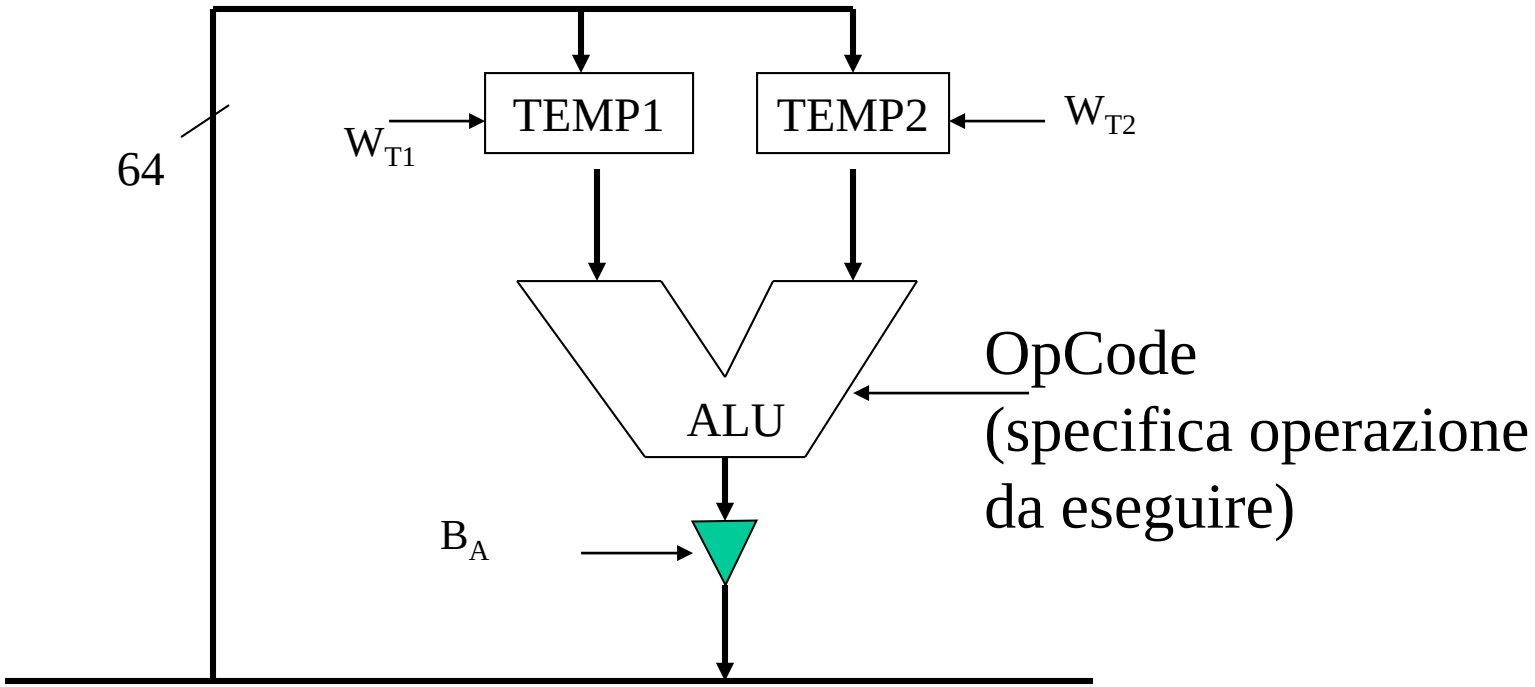


Switch dell'identificativo dei registri

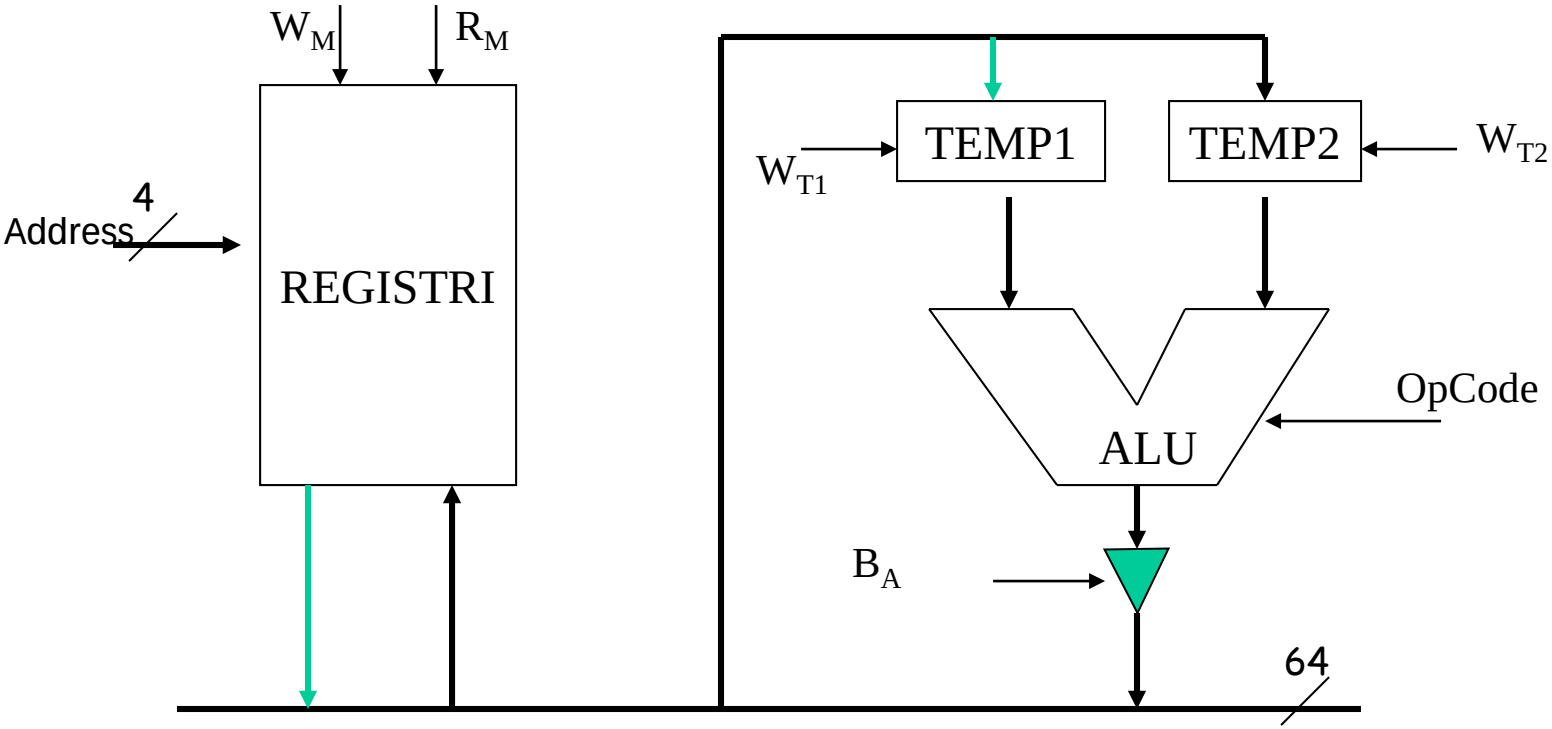


z64- ALU

- Esegue le operazioni aritmetiche e logiche dei valori memorizzati in due registri tampone (non visibili al programmatore) Temp1 e Temp2
- Il risultato è posto in un registro generale Ri

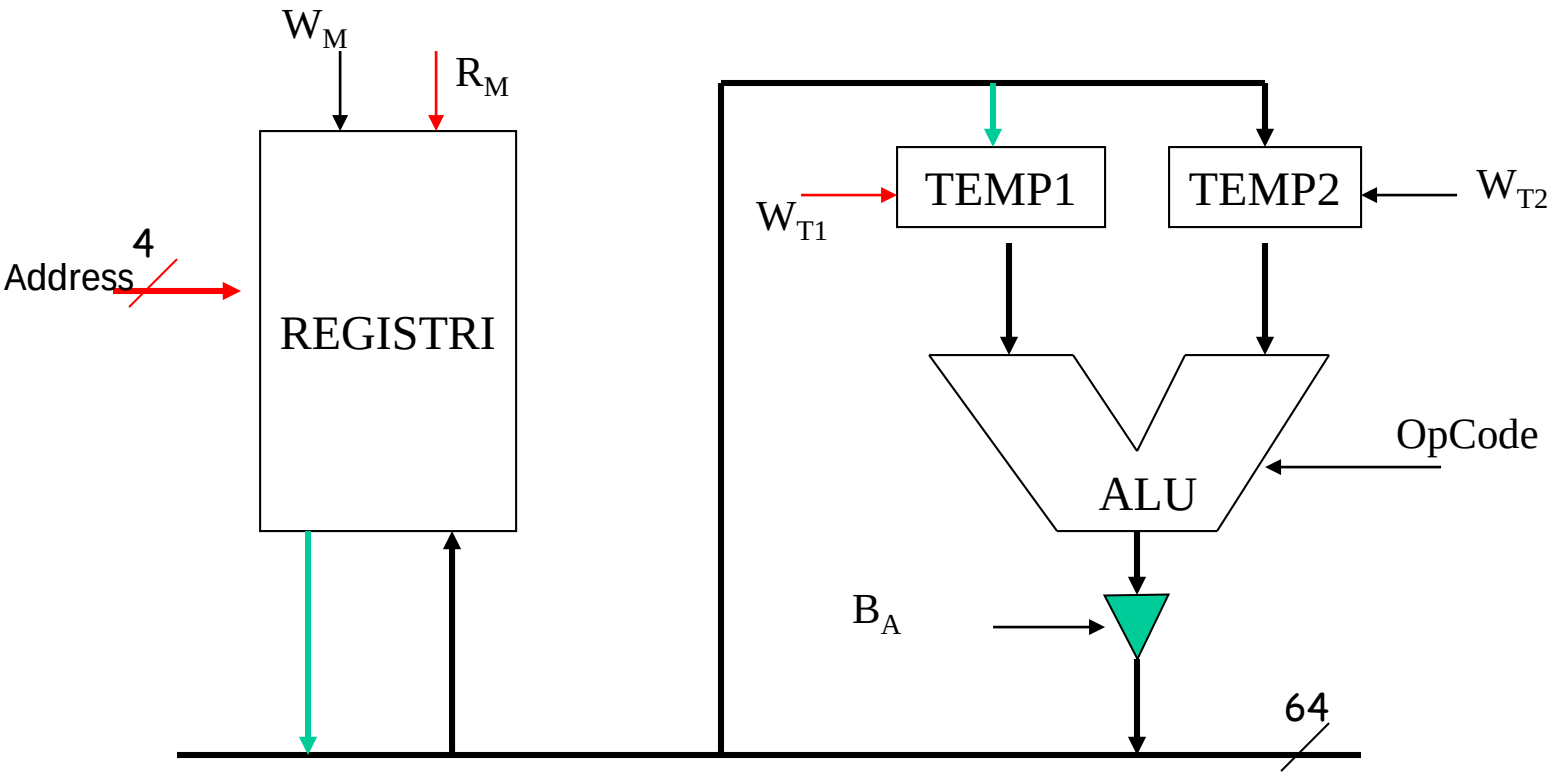


z64 - ALU, esempio: esecuzione addq R8,R15



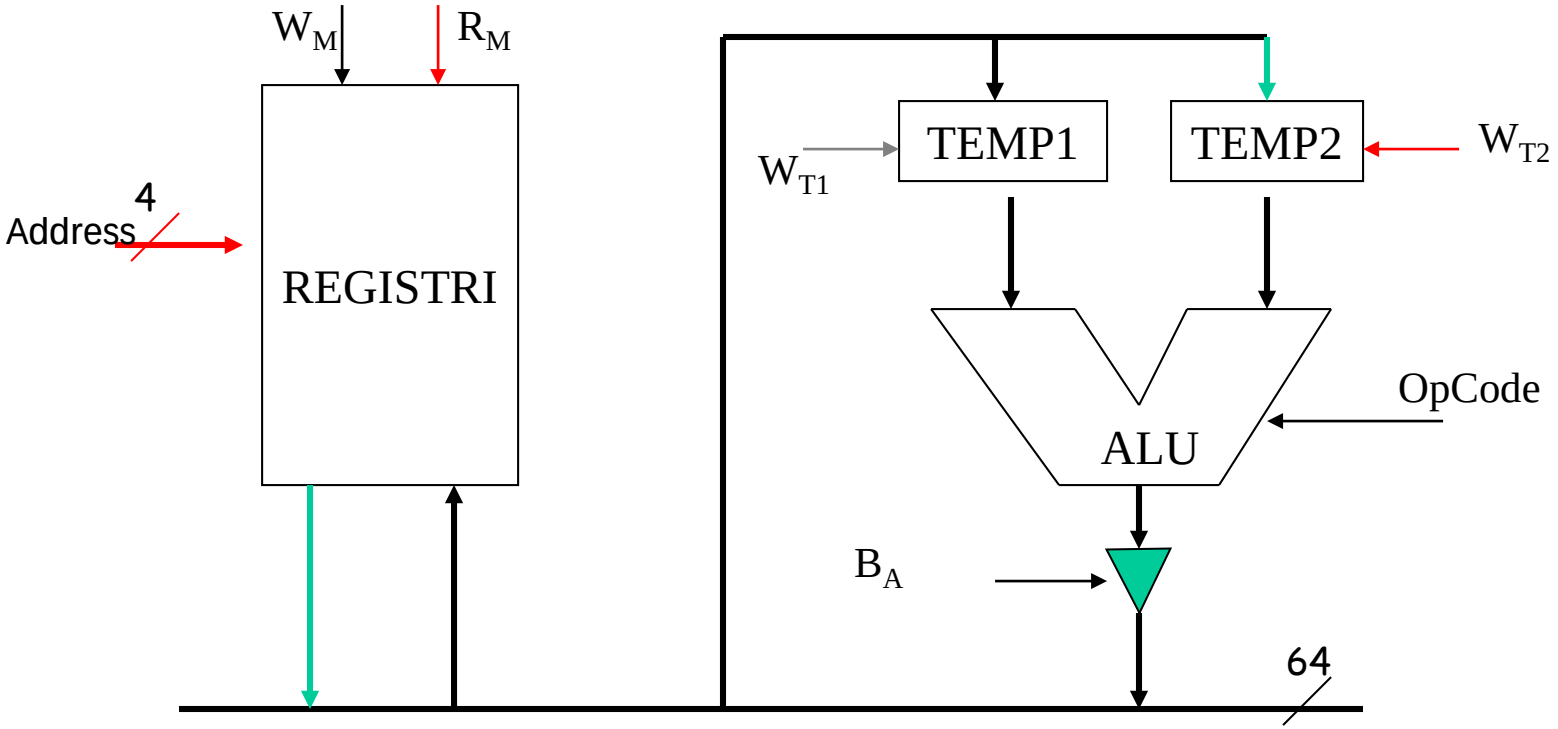
z64 - ALU, esempio: esecuzione addq R8,R15

1. R8 -> Temp1 R_M=1, Address = 1000, W_{T1}=1



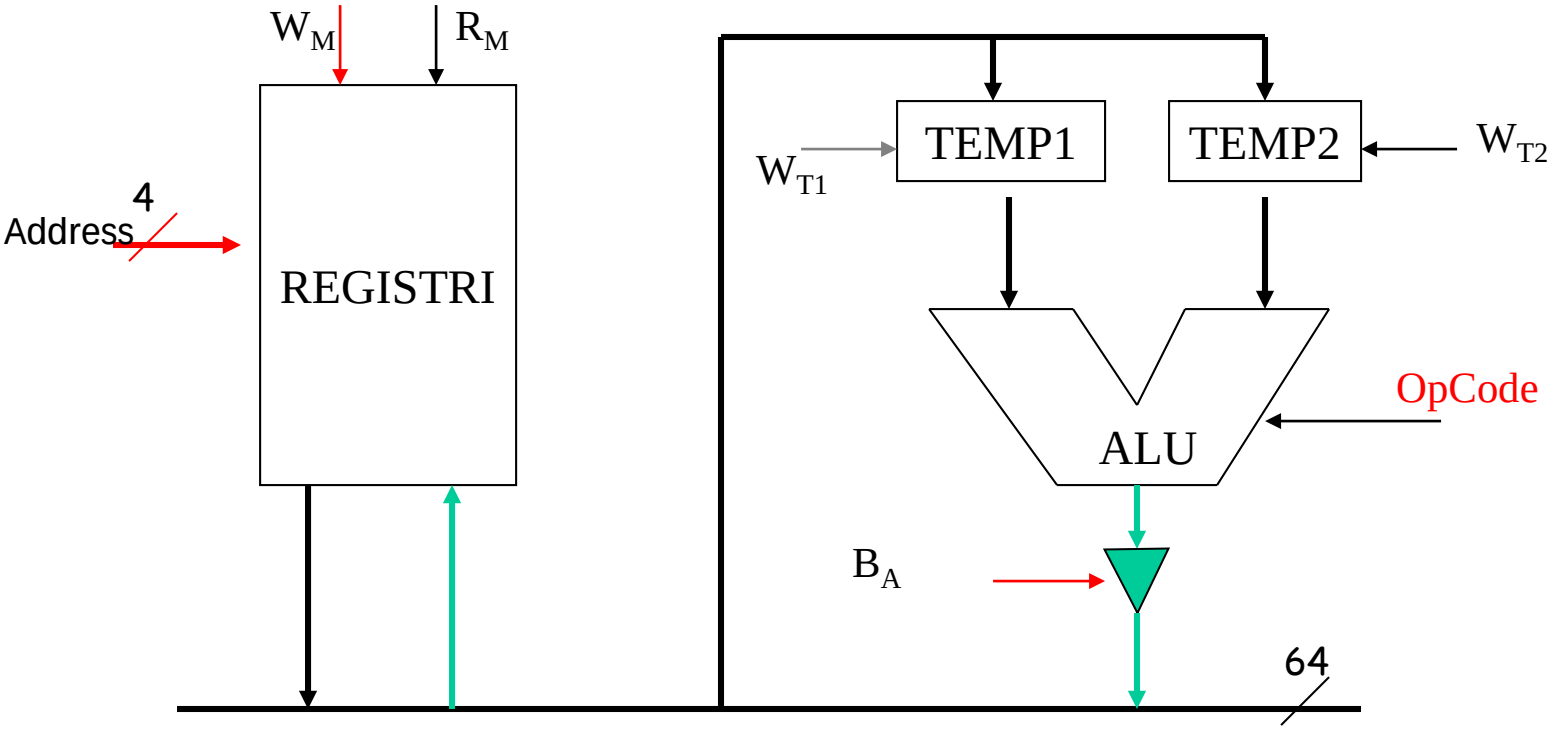
z64 - ALU, esempio: esecuzione addq R8,R15

1. R8 -> Temp1 R_M=1, Address = 1000, W_{T1} = 1
2. R15 -> Temp2 R_M=1, Address = 1111, W_{T2} = 1



z64 - ALU, esempio: esecuzione addw R2,R1

1. R8 -> Temp1 R_M=1, Address = 1000, W_{T1} = 1
2. **R15 -> Temp2 R_M=1, Address = 1111, W_{T2} = 1**
3. **ALU-OUT(Temp1+Temp2)->R15**
   W_M=1, Address = 1111, OpCode = addw, B_A=1

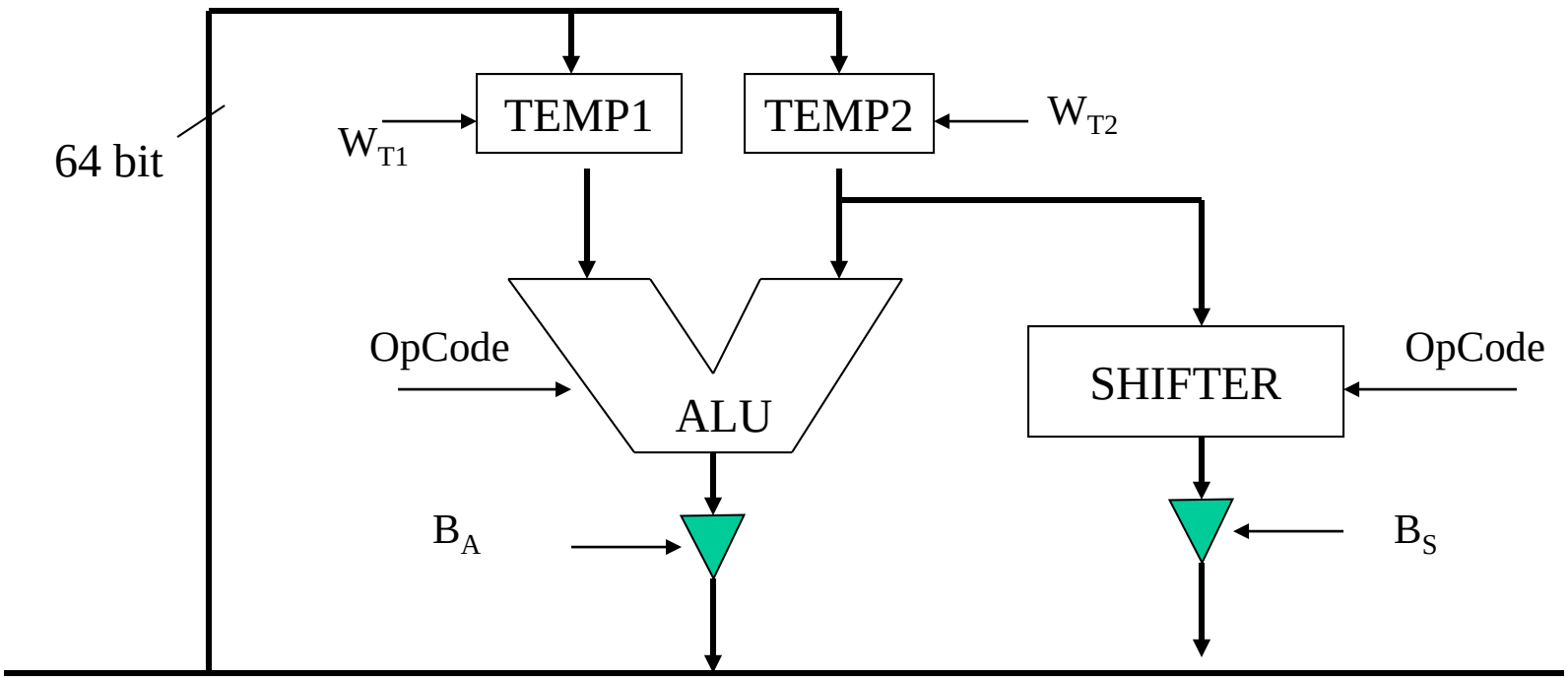


Osservazioni

- Per l'esecuzione dell'istruzione (senza considerare la fase di fetch) sono state necessarie 3 operazioni elementari
- Ogni operazione viene eseguita in un ciclo di clock
- In generale il numero di cicli di clock richiesti per completare una istruzione è variabile e dipende dall'istruzione. Tale paramentro viene indicato con CPI (Clock per Instruction)
- La velocità di esecuzione di un programma dipende anche dal numero medio di CPI

z64 - Shifter

Usato per eseguire operazioni di scorrimento di posizioni, nonché per lo spostamento di dati tra registri interni (i registri tampone non possono scrivere sul bus mentre i segnali di controllo valgono per tutti i registri)



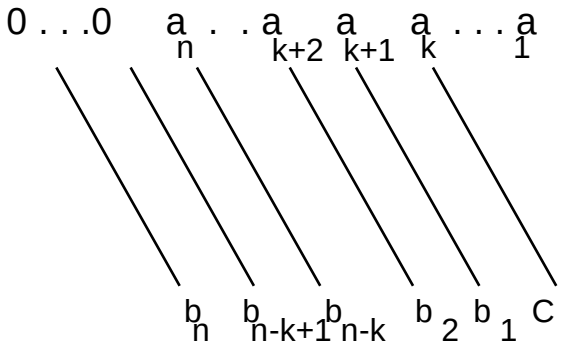
Shifter (background)

Spostamento logico a destra di k posti.

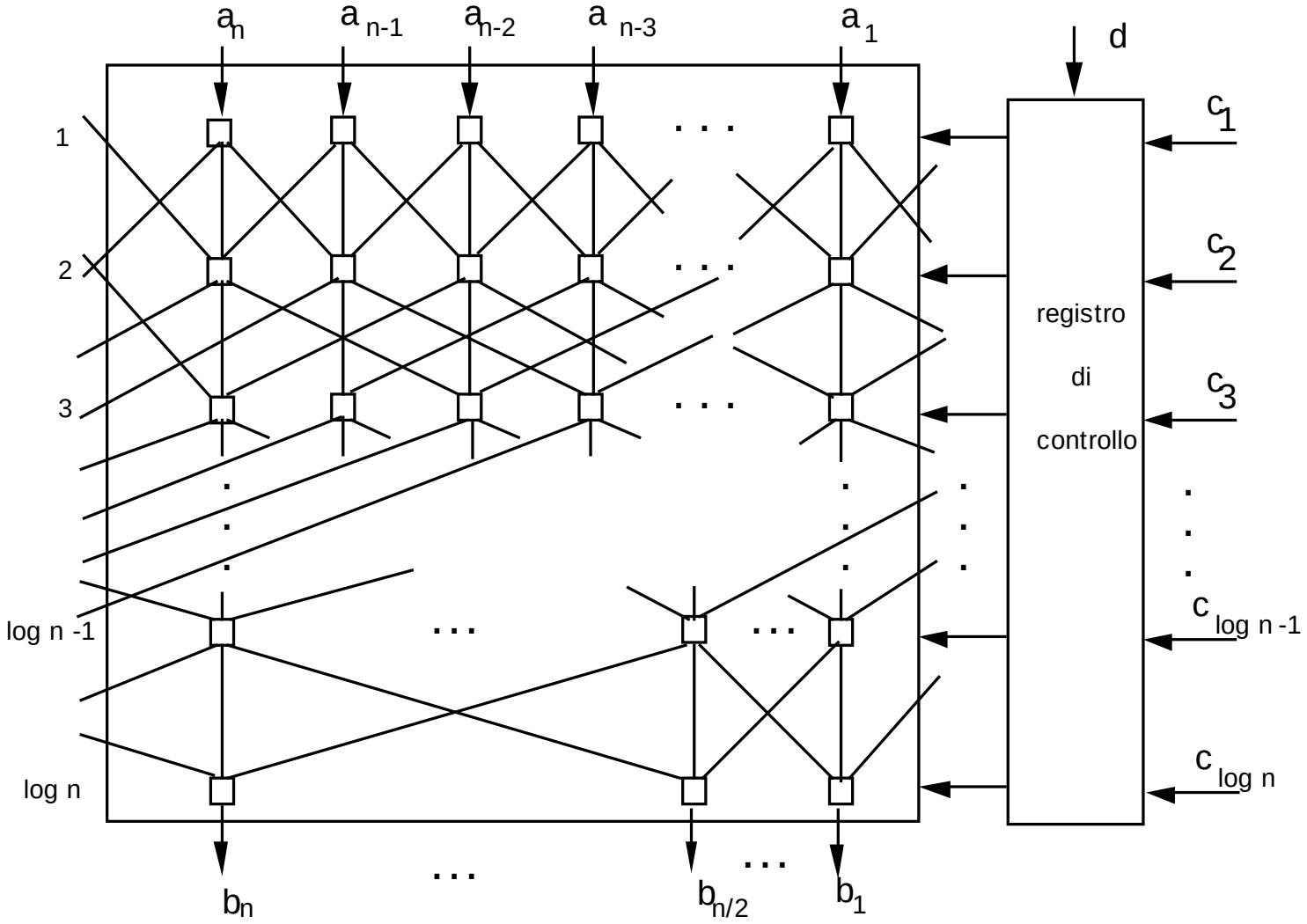
$bn\text{-}i = 0 \quad (per\ 0 \leq i < k)$

$bi = ai+k\ (per\ 1 \leq i \leq n - k)$

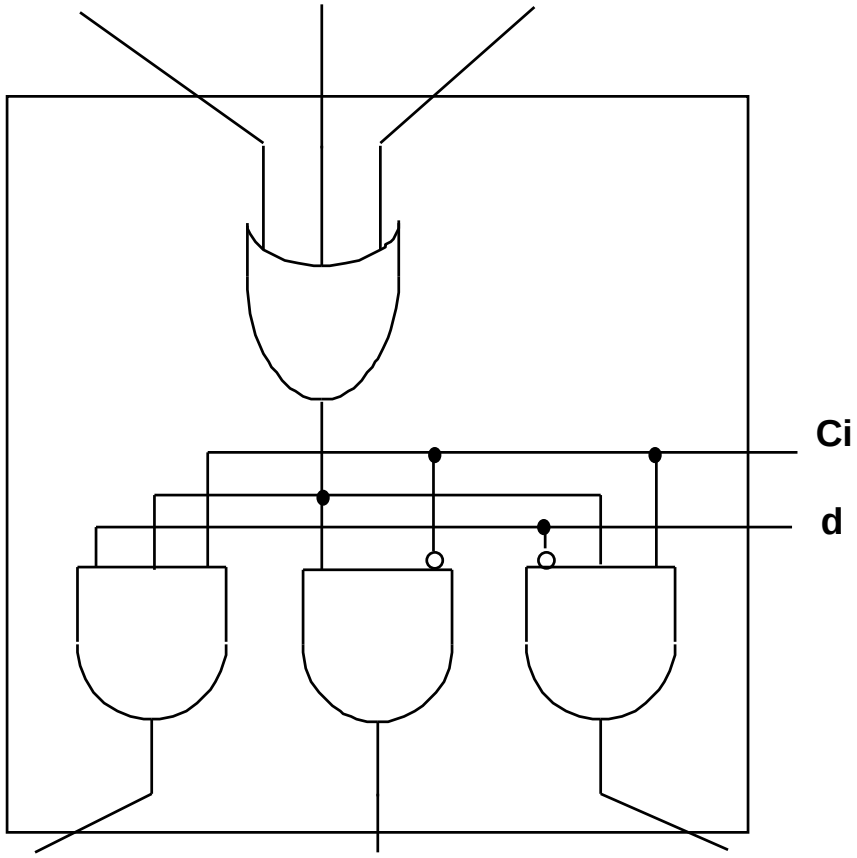
$C = ak$



Barrel shifter

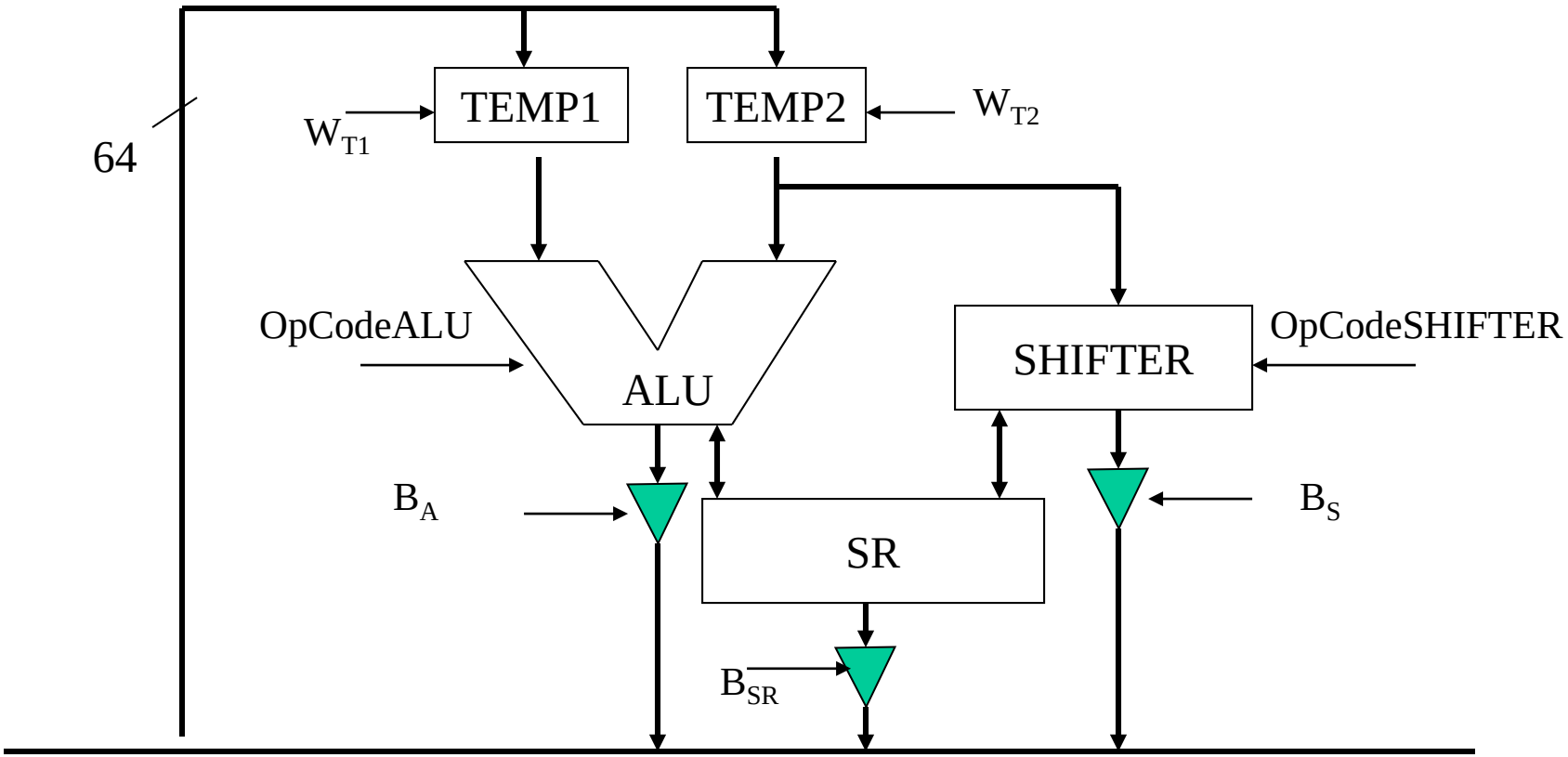


Schema di una cella



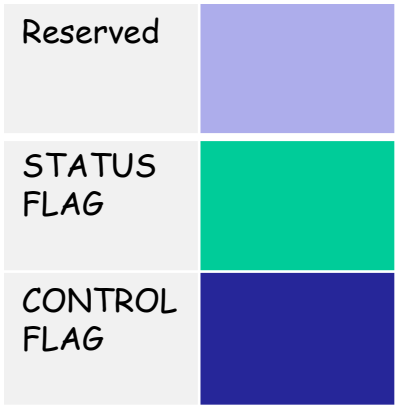
z64 - Status Register

Contiene informazioni sull'esito dell'ultima operazione (ex. zero, overflow). Usato anche come ingresso per alcune operazioni (ex. Salti condizionati)



z64 - Status Register

OF: Overflow Flag
DI: Direction Flag
IF: Interrupt Flag
SF: Sign Flag
ZF: Zero Flag
PF: Parity Flag
CF: Carry Flag

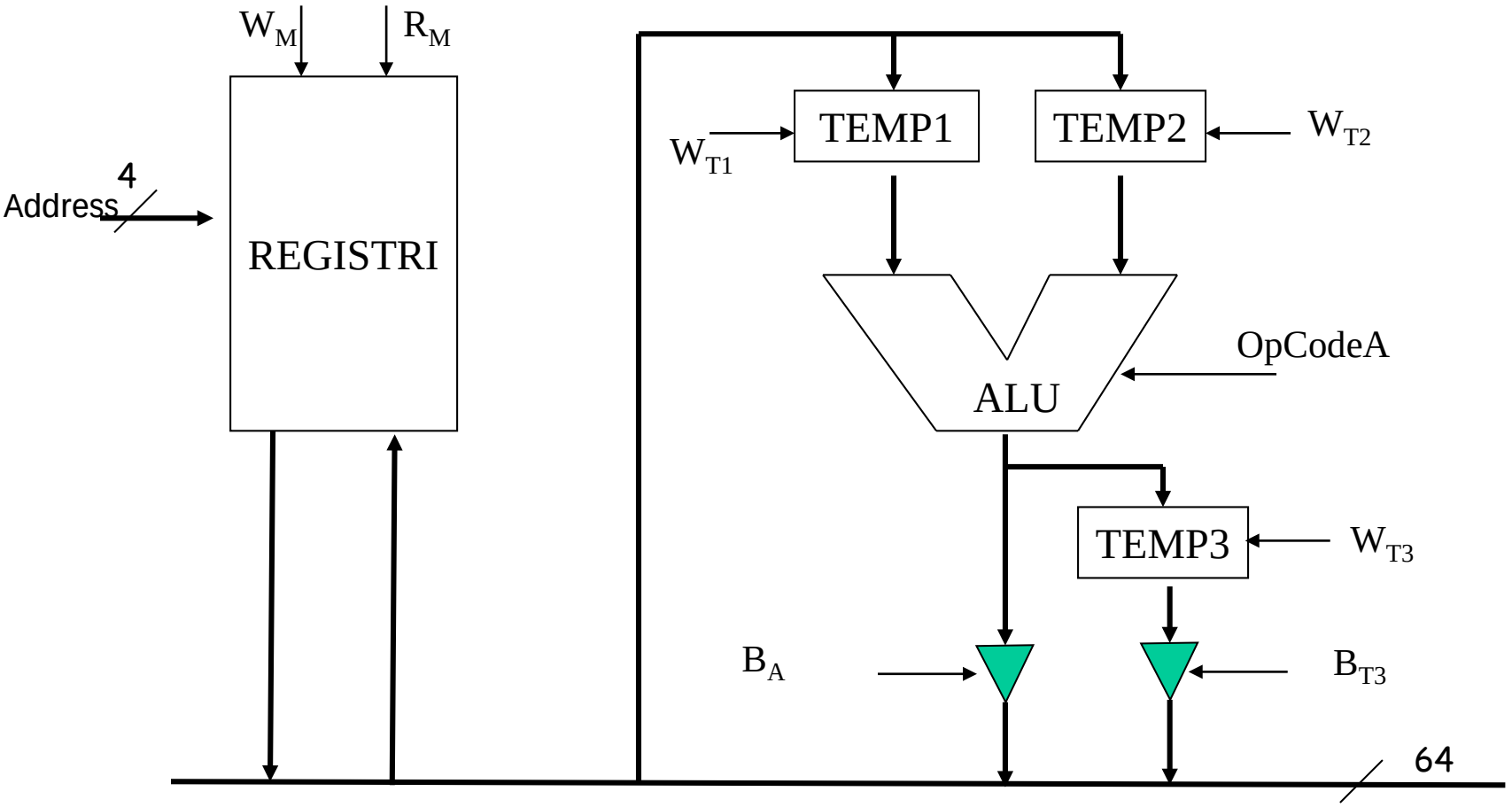


esecuzione
addq 200(R9,R10,4), R8

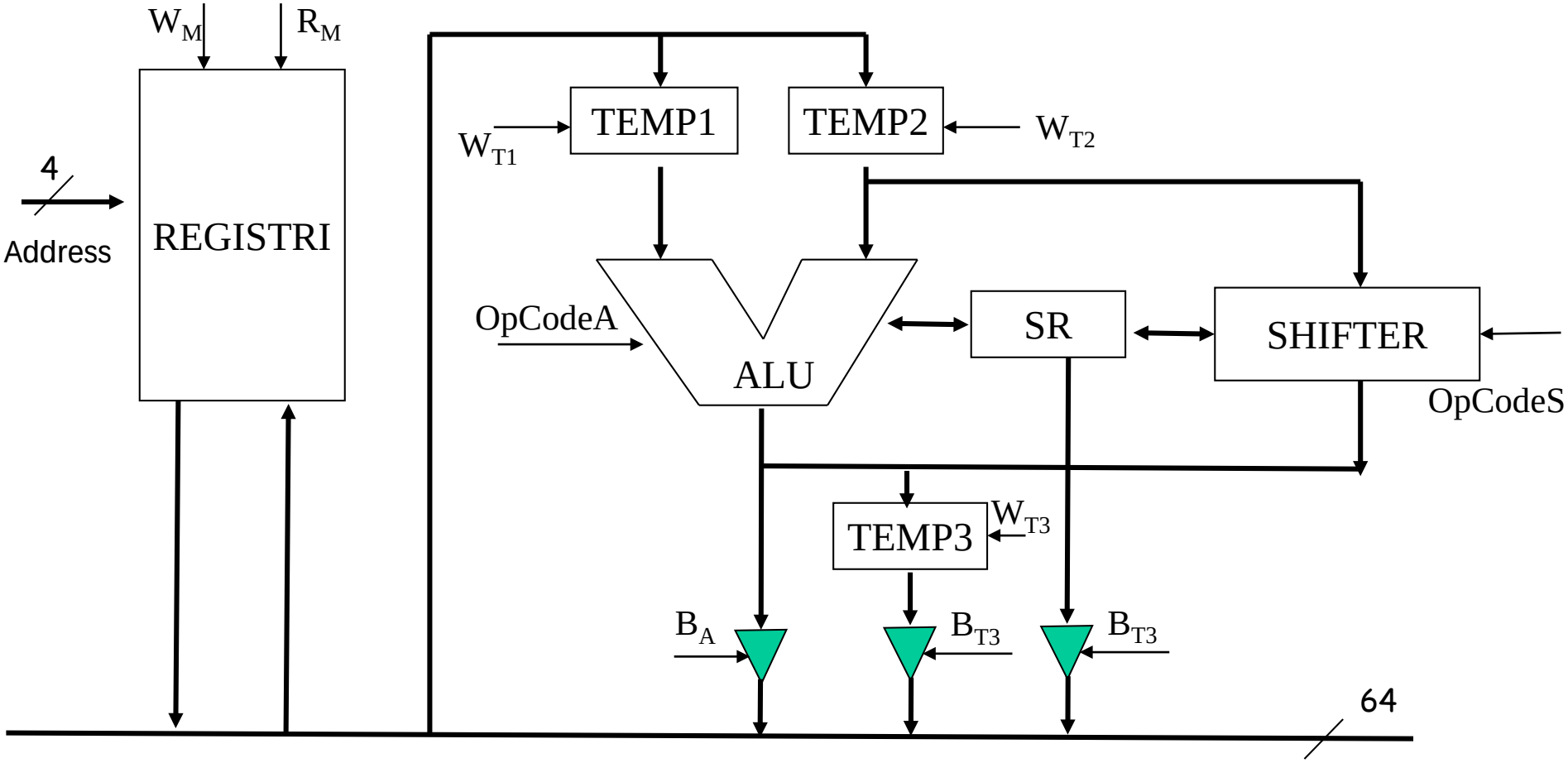
- Somma
  - il contenuto della locazione di memoria il cui indirizzo è dato dalla somma di 200, con il contenuto del registro R9 e con quello del registro R10 moltiplicato di 4 (shiftato di 2)
  - con il contenuto del registro R8
- e metti il risultato in R8

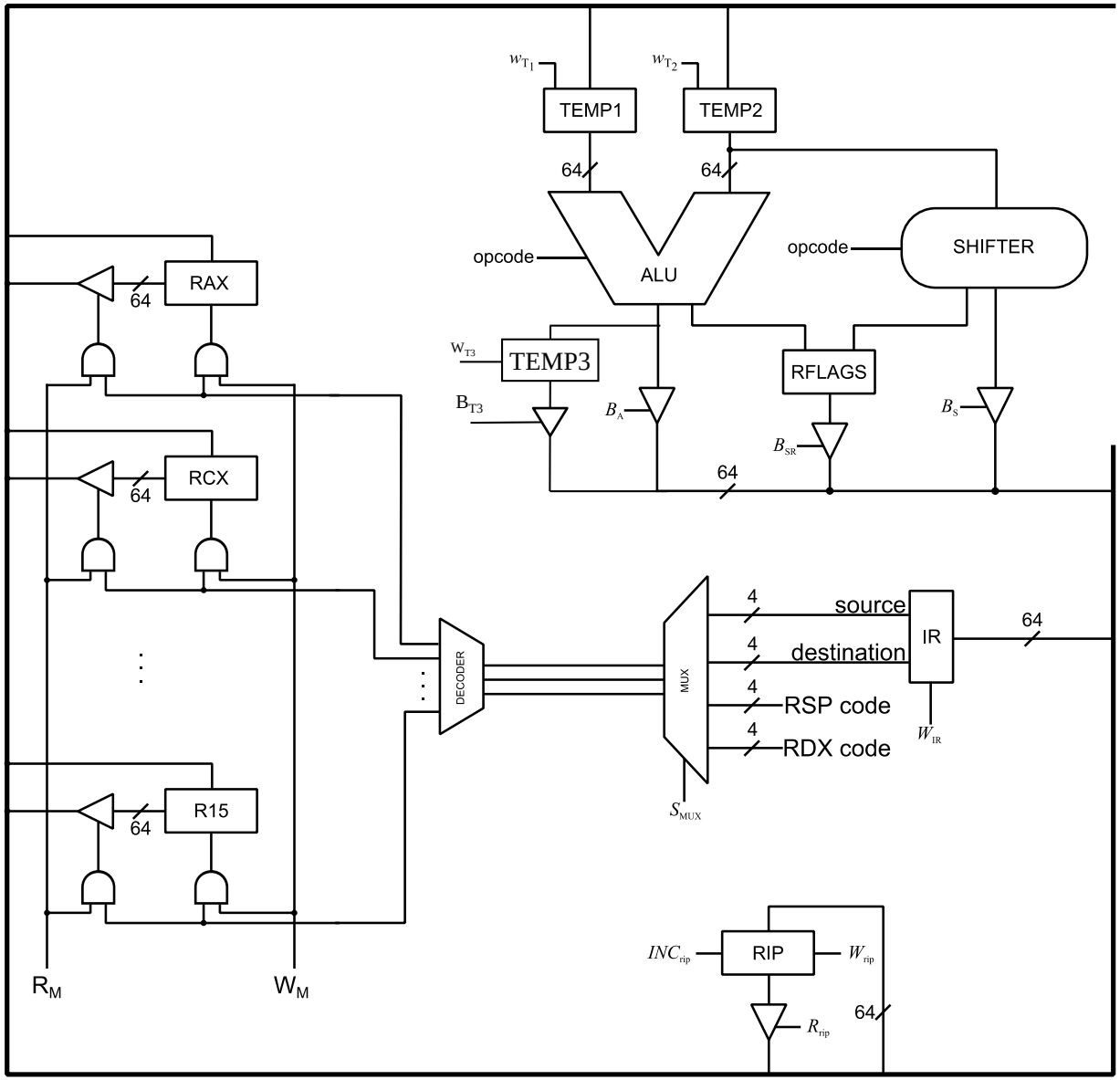
NECESSITA' DI UN ALTRO REGISTRO TAMPONE

Aggiunta registro temporaneo per eseguire istruzioni complesse



Architettura con tre registri TEMP



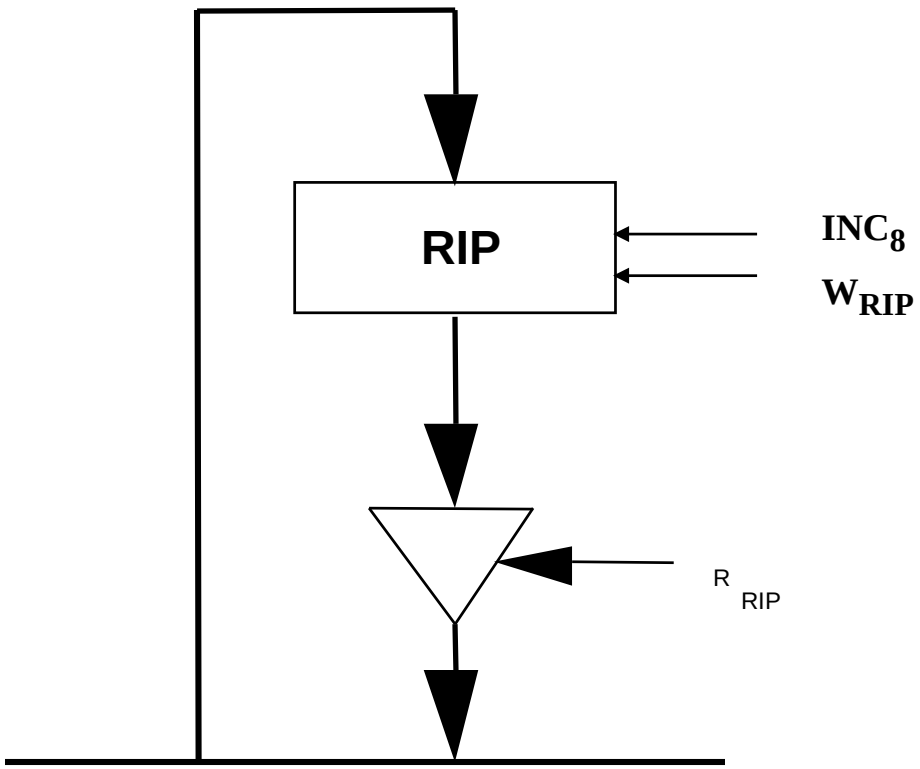


singola linea

linee multiple

N.B. non sono evidenziate le variabili di condizione che da SR e IR vanno al SCO

Incremento RIP

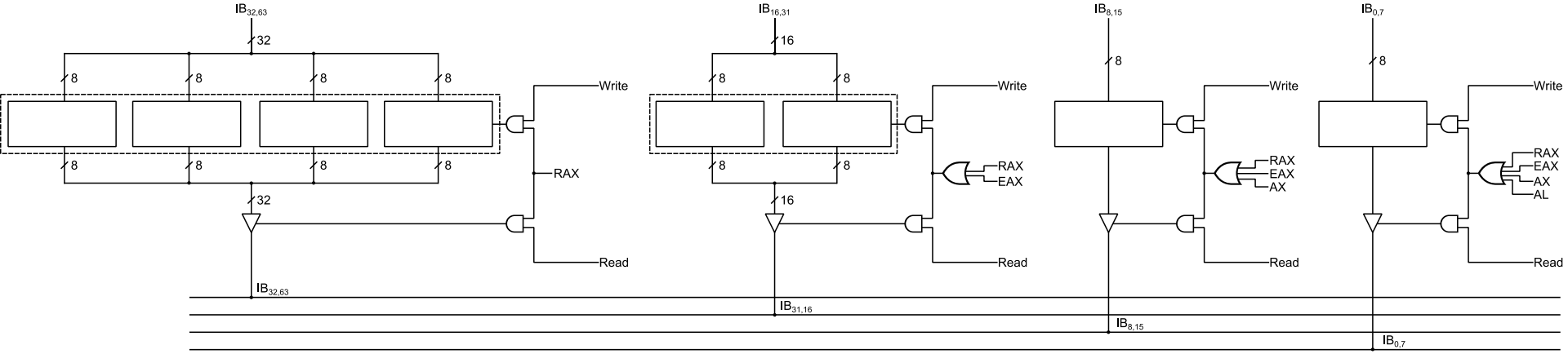


Il RIPdeve essere incrementato (se non si eseguono istruzione di salto)

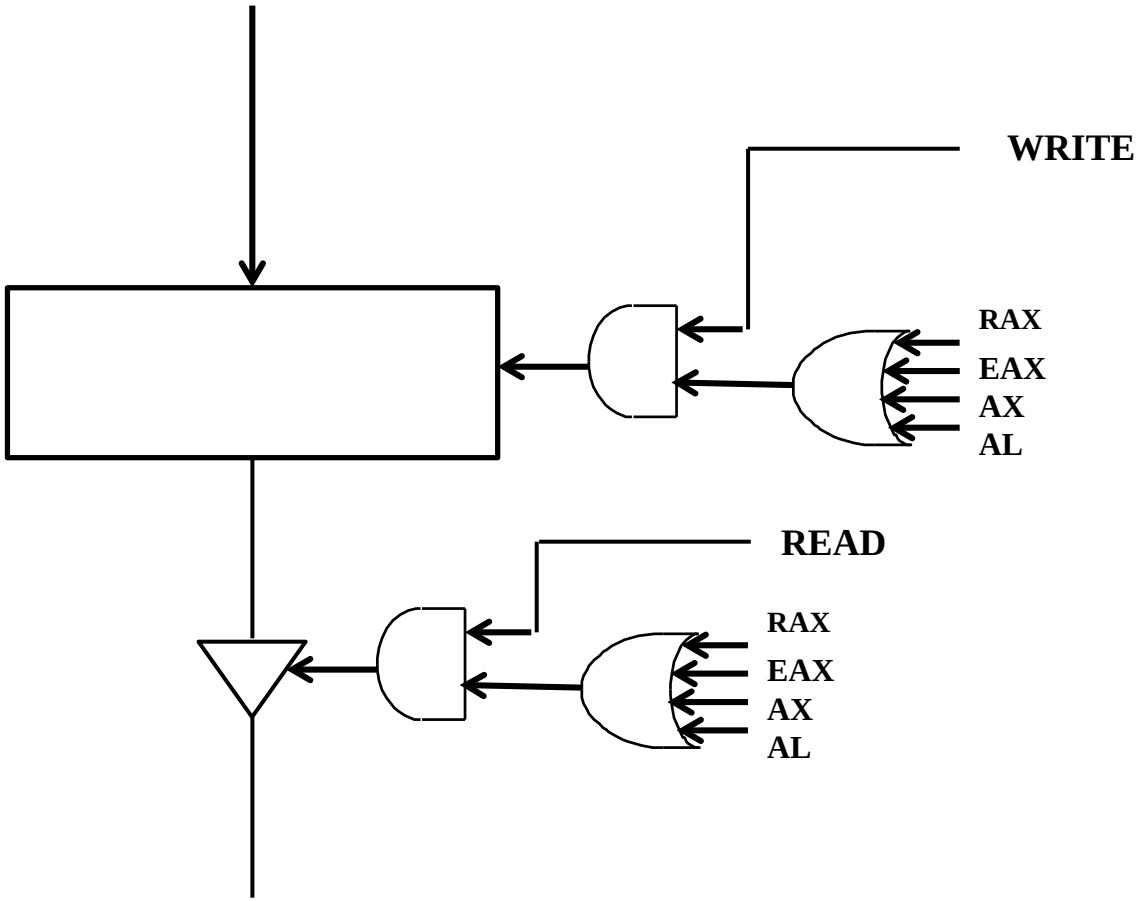
NOTA: le istruzioni z64 possono essere solo di 64 o 128 bit, quindi è sufficiente incrementare di 8 il RIP

Modifica del banco dei registri per accedere a dati a formato variabile

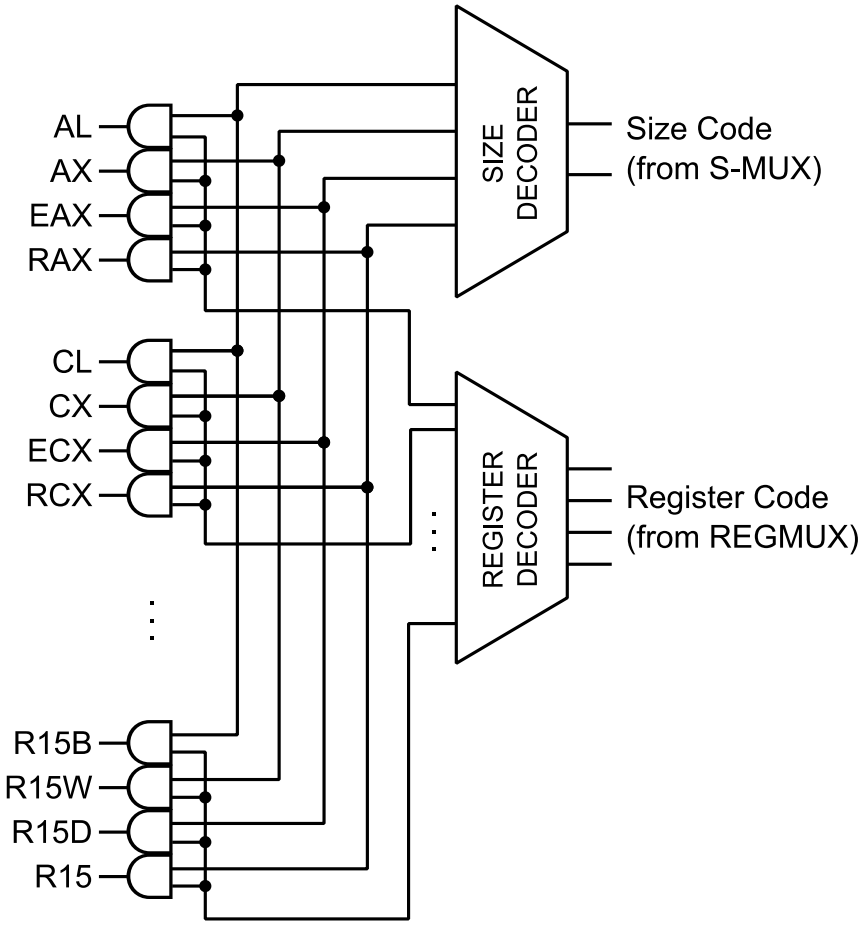
registri RAX, EAX, AX, AL



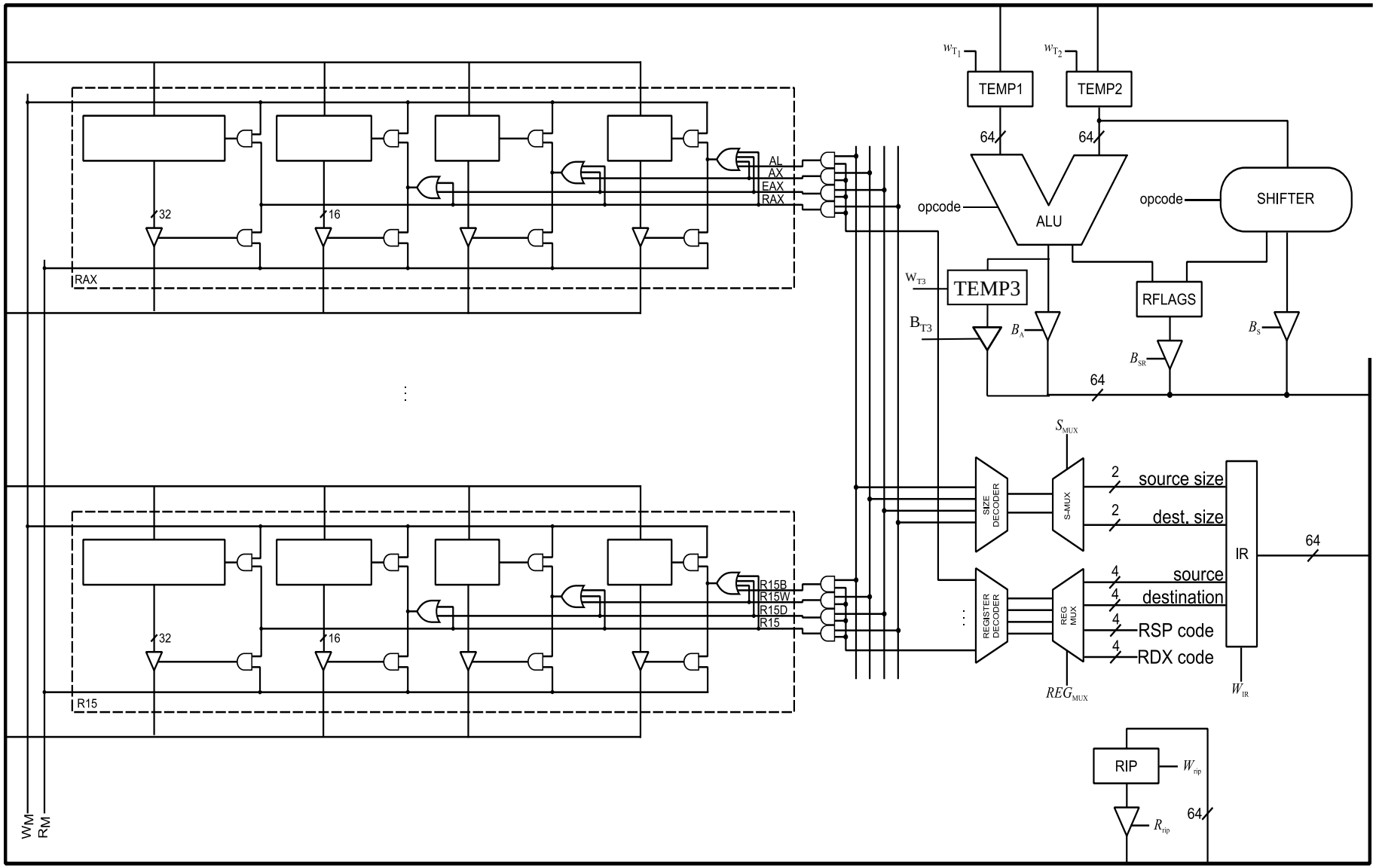
Espansione
registro AL



Modalità di selezione dei registri



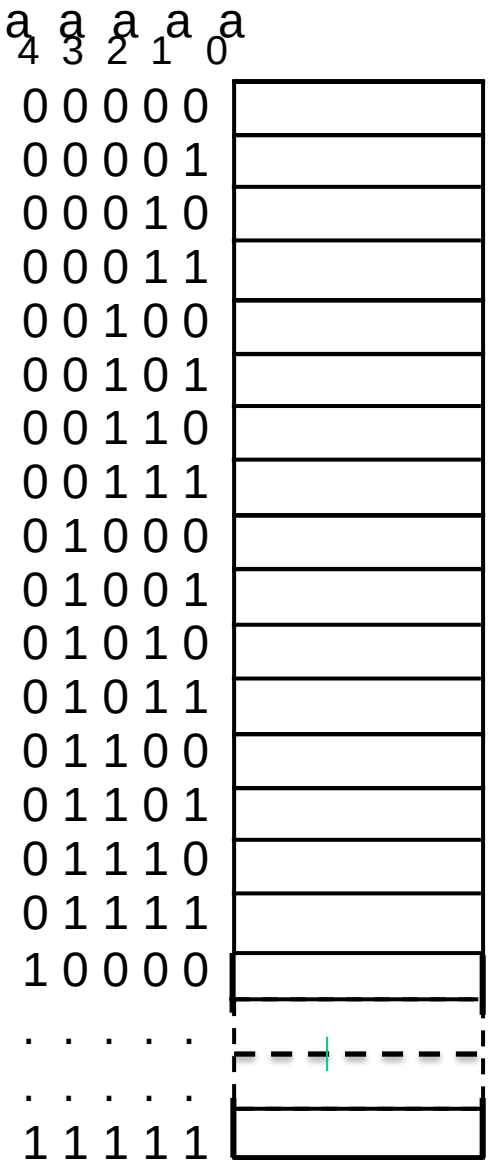
Architettura senza interfacce



z64 - Interazione con la memoria

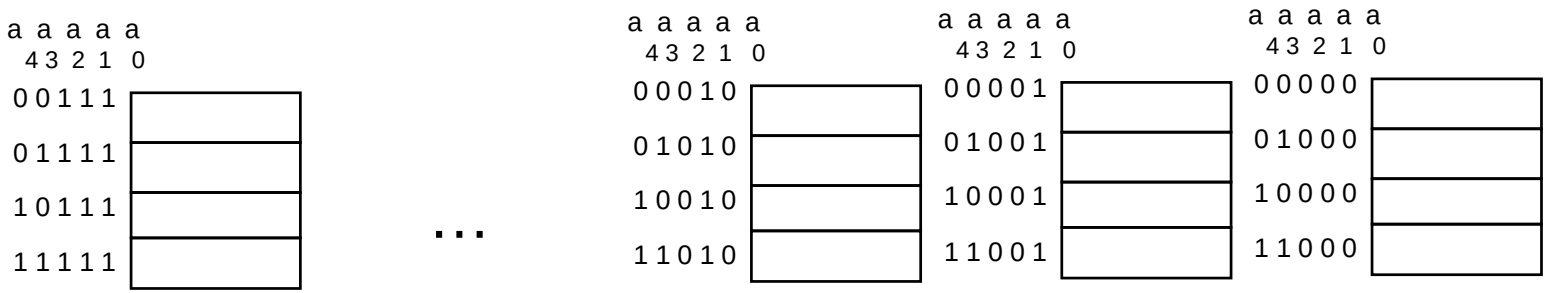
- La memoria contiene sia i dati che le istruzioni e può essere sia letta che scritta.
- E' necessario quindi:
  - Prelevare istruzioni
  - Leggere dati
  - Scrivere dati
- Le operazioni di lettura/scrittura avvengono fra una locazione di memoria e un registro (registro del banco registri, IR, RIP, TEMP1,TEMP2 e TEMP3)
- E' necessario quindi instradare opportunamente i dati ricevuti dalla memoria verso i registri e viceversa.

Memoria: organizzazione logica



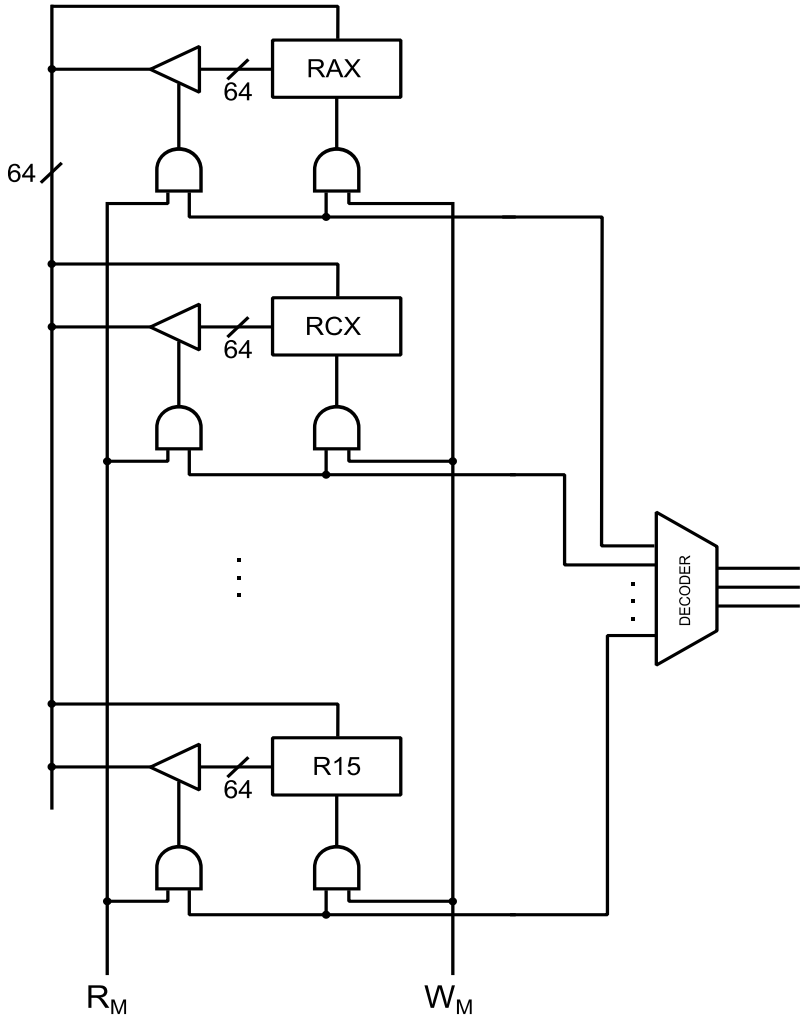
Organizzazione logica a vettore di 32 celle di memoria

Organizzazione logica di 32 celle di memoria con 8 moduli di 8 bit ciascuno

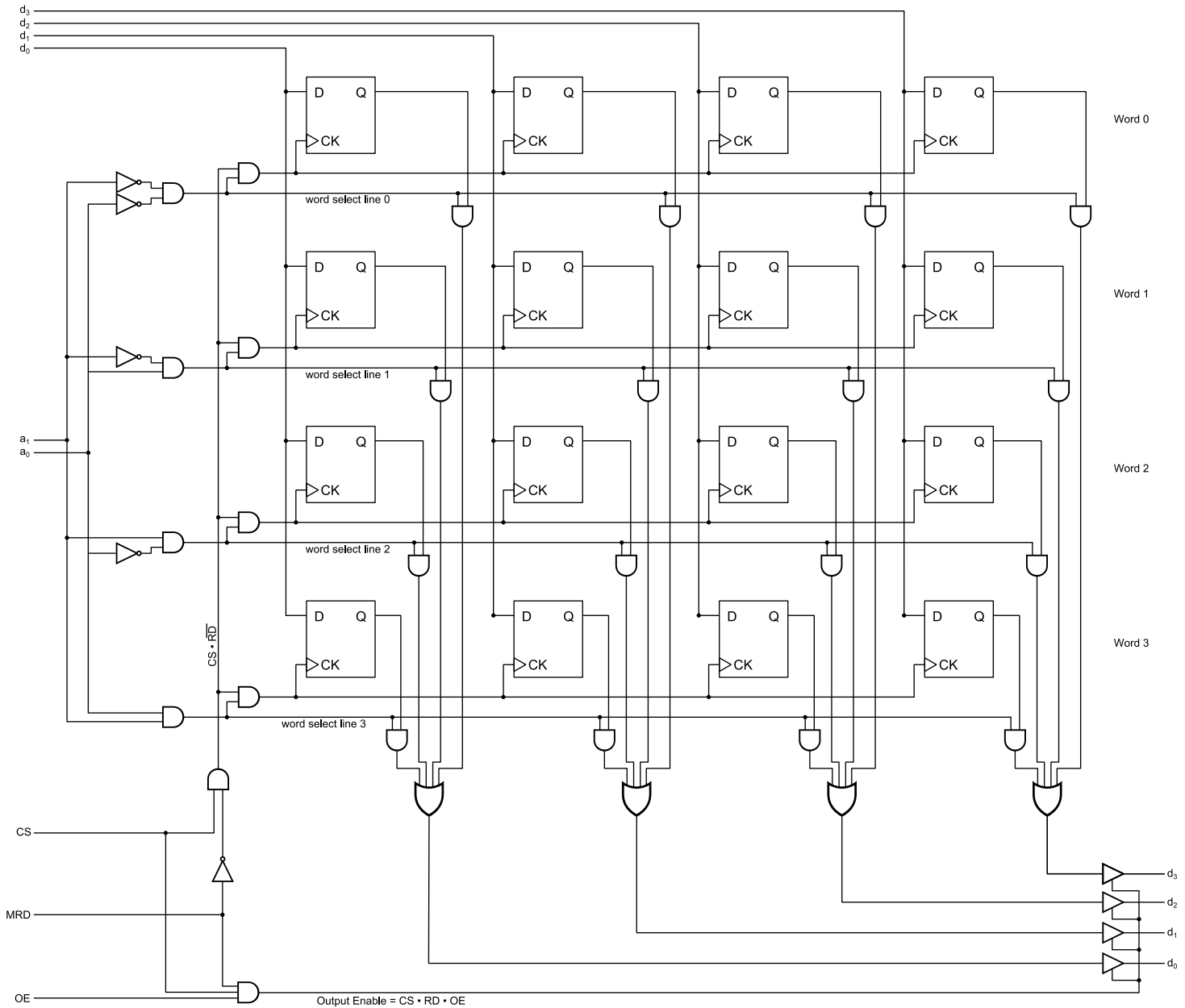


I tre bit meno significativi identificano il modulo, quelli più significativi la riga

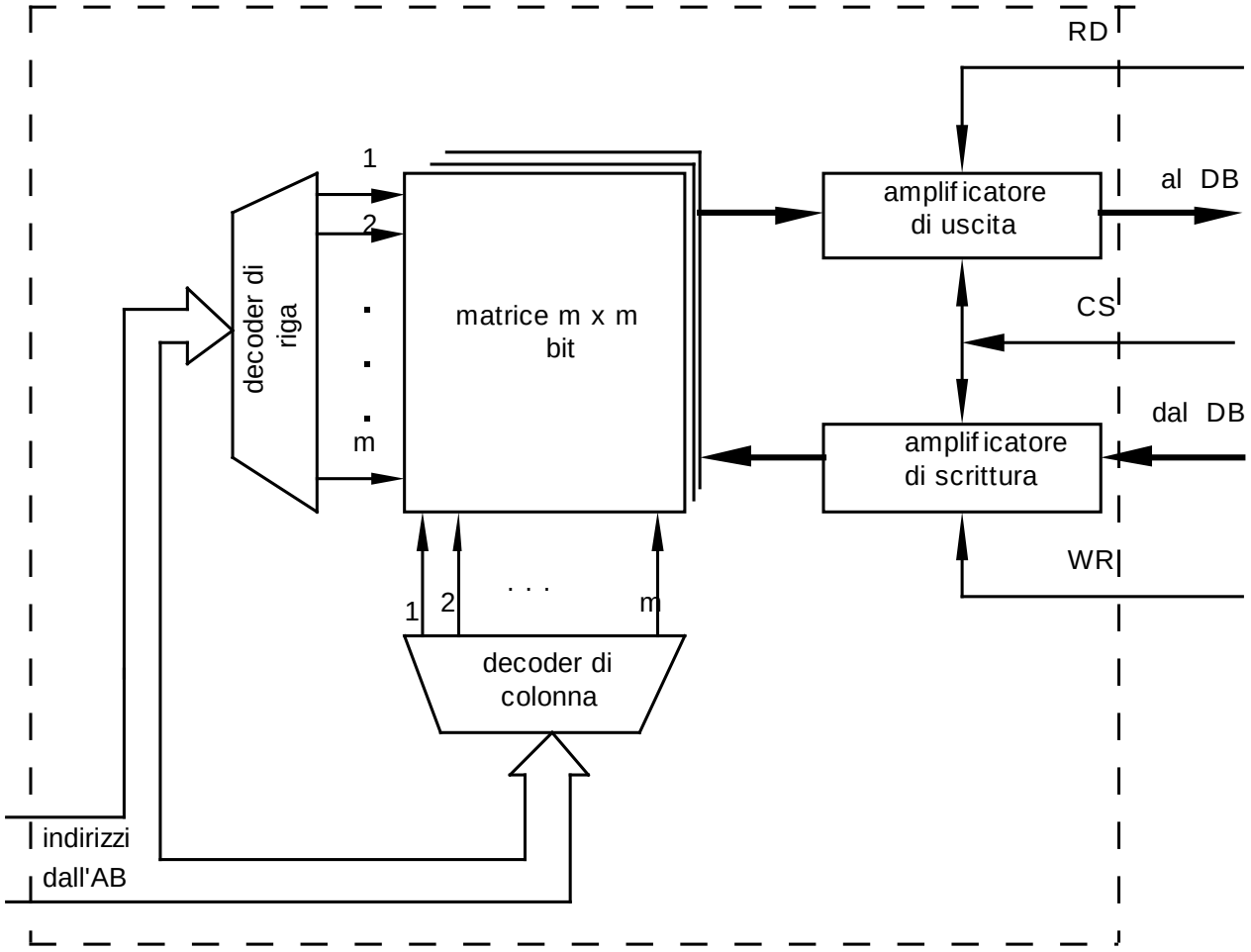
Architettura di un modulo di memoria, come banco di registri



Organizzazione di base di una memoria RAM statica



Memoria RAM statica



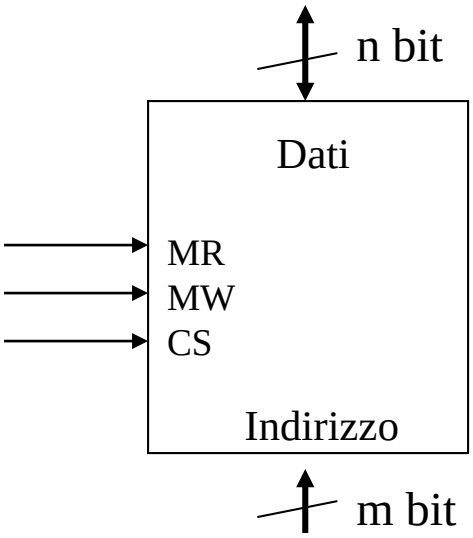
linea singola

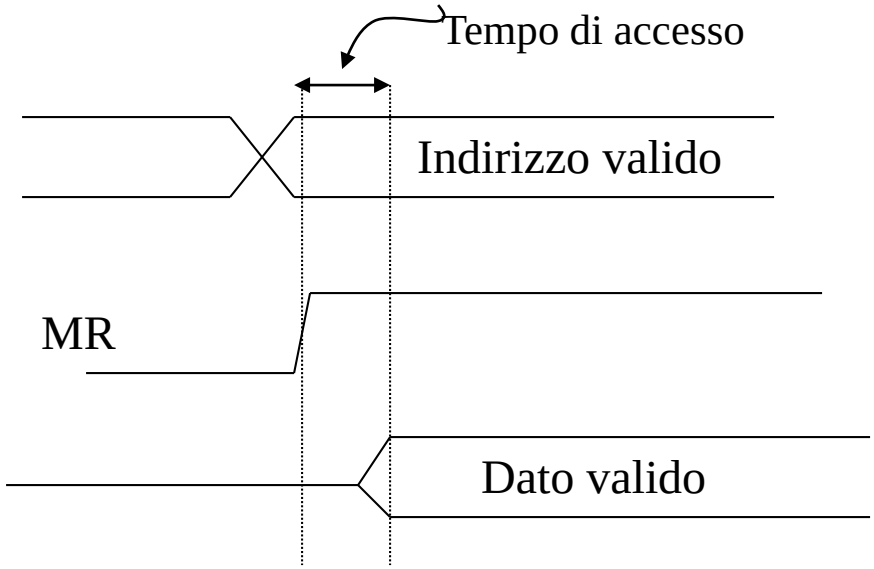
linee multiple

Memoria comportamento esterno

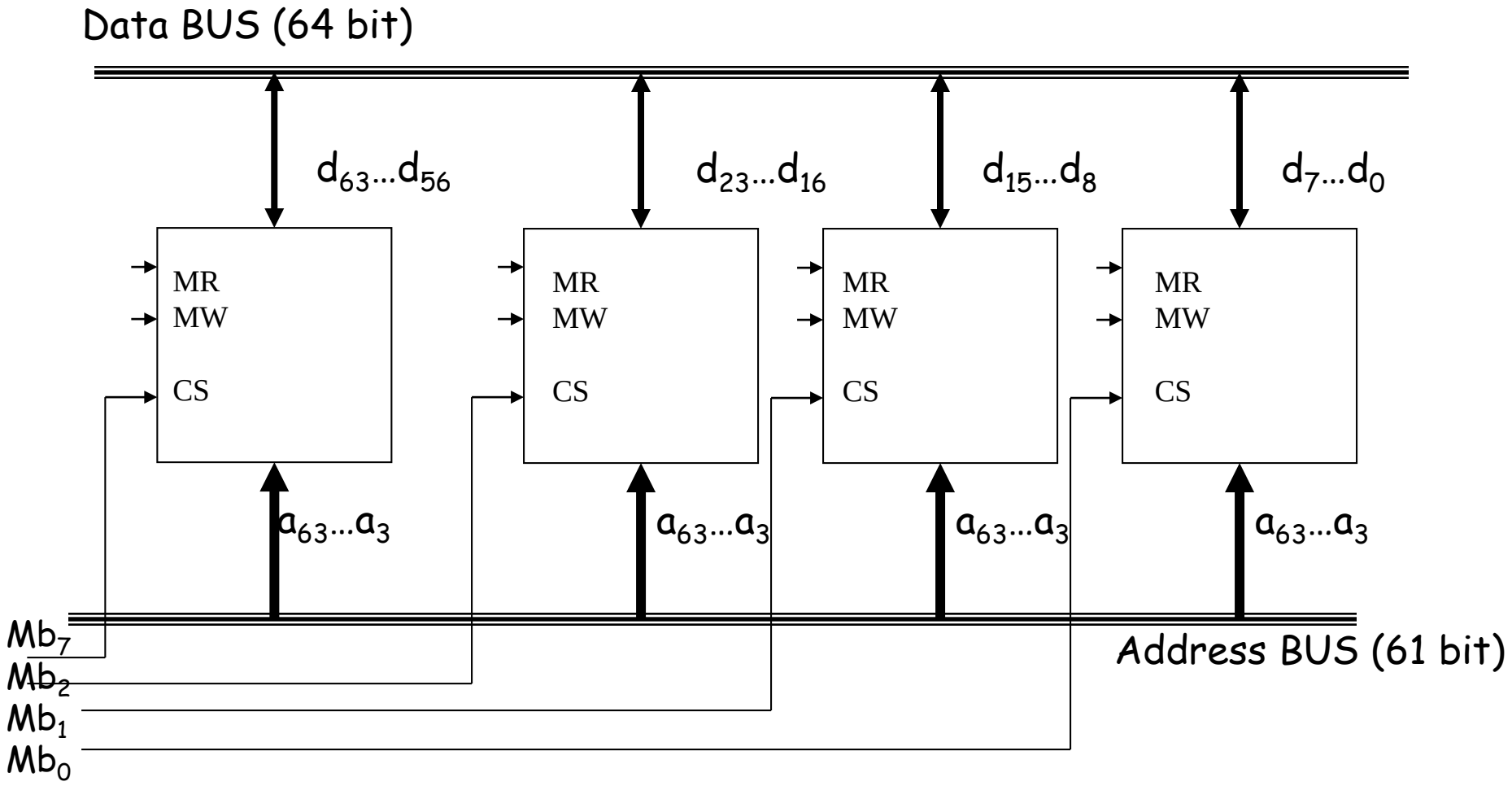
Funzionalmente è caratterizzata dai seguenti segnali

- Indirizzo della parola da leggere/scrivere
- MR, affermato se si vuole leggere
- MW, affermato se si vuole scrivere
- CS, Abilita l'intero modulo (Chip Select)
- Dati



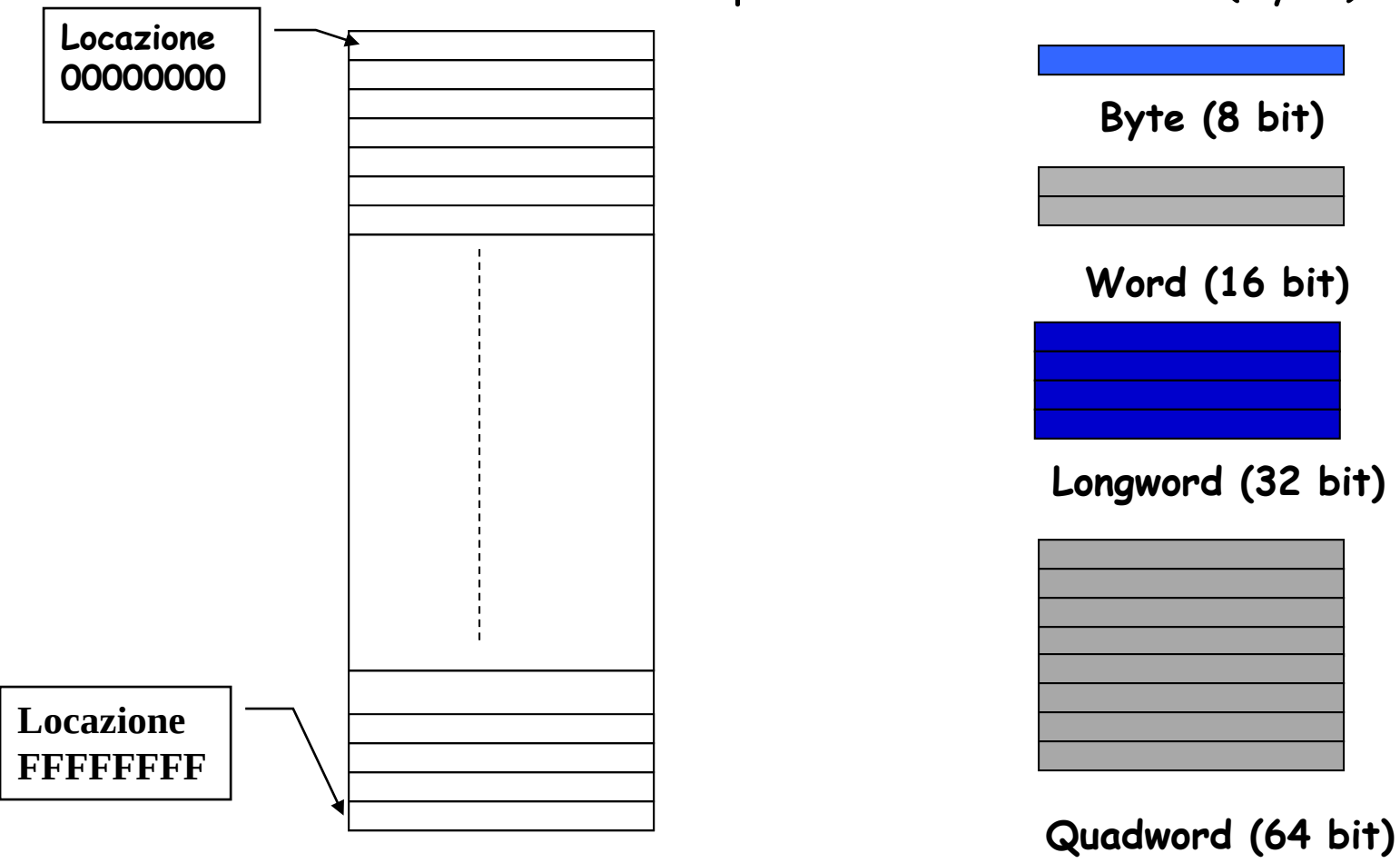


Memoria: organizzazione a moduli

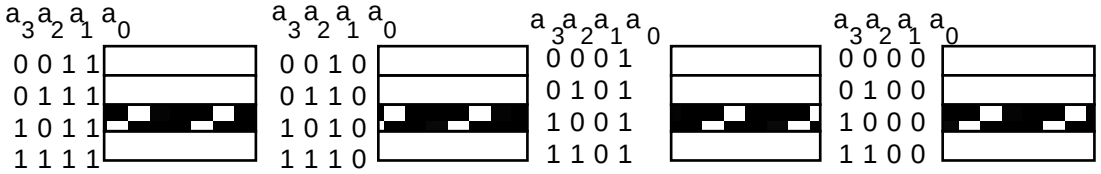


Memoria: spazio di indirizzamento

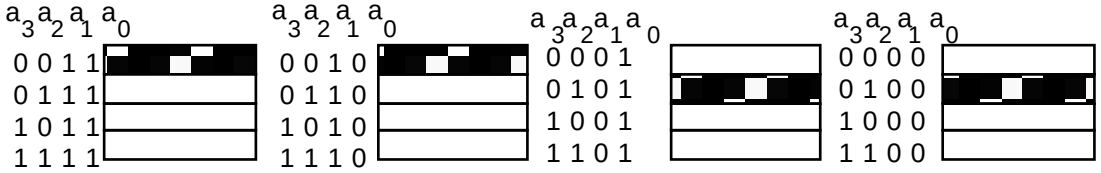
Lo spazio di indirizzamento del Y64 e' monodimensionale e
Composto da 2^{64} locazioni (byte)



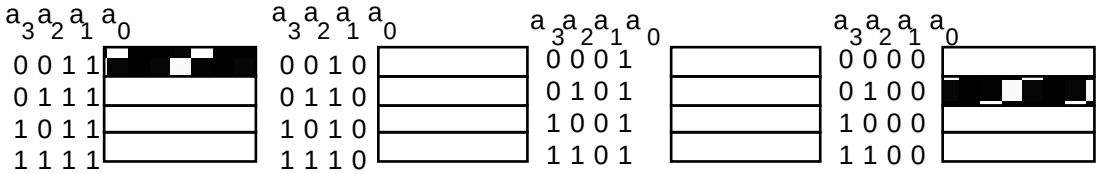
Memoria: allineamento e disallineamento (memoria a 32 bit)



Esempio di memorizzazione di una informazione di quattro byte allineati sullo stesso indirizzo di riga.

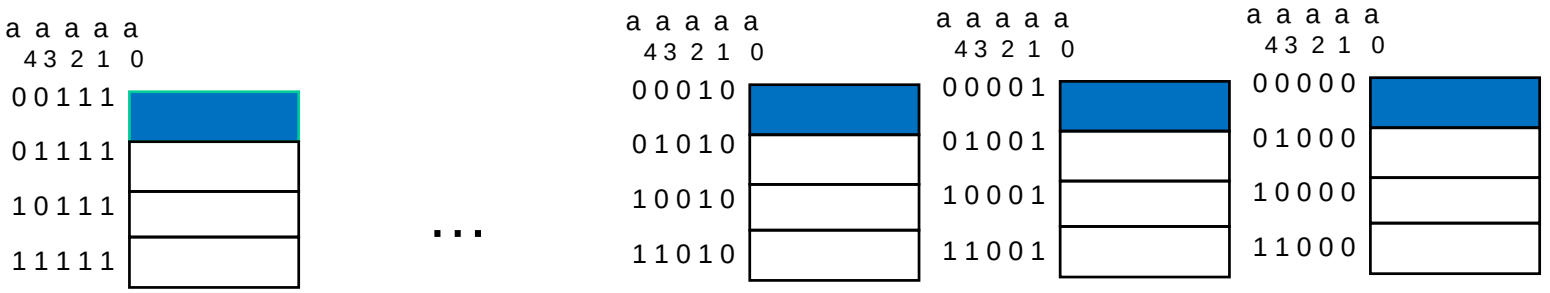


Esempio di memorizzazione di una informazione di quattro byte disallineati

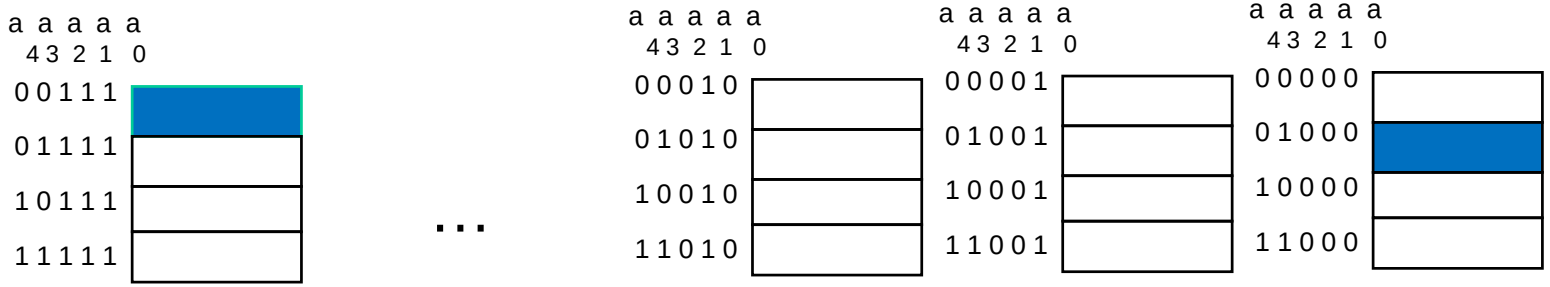


Esempio di memorizzazione di una informazione di due byte disallineati

Memoria: allineamento e disallineamento (memoria a 64 bit)

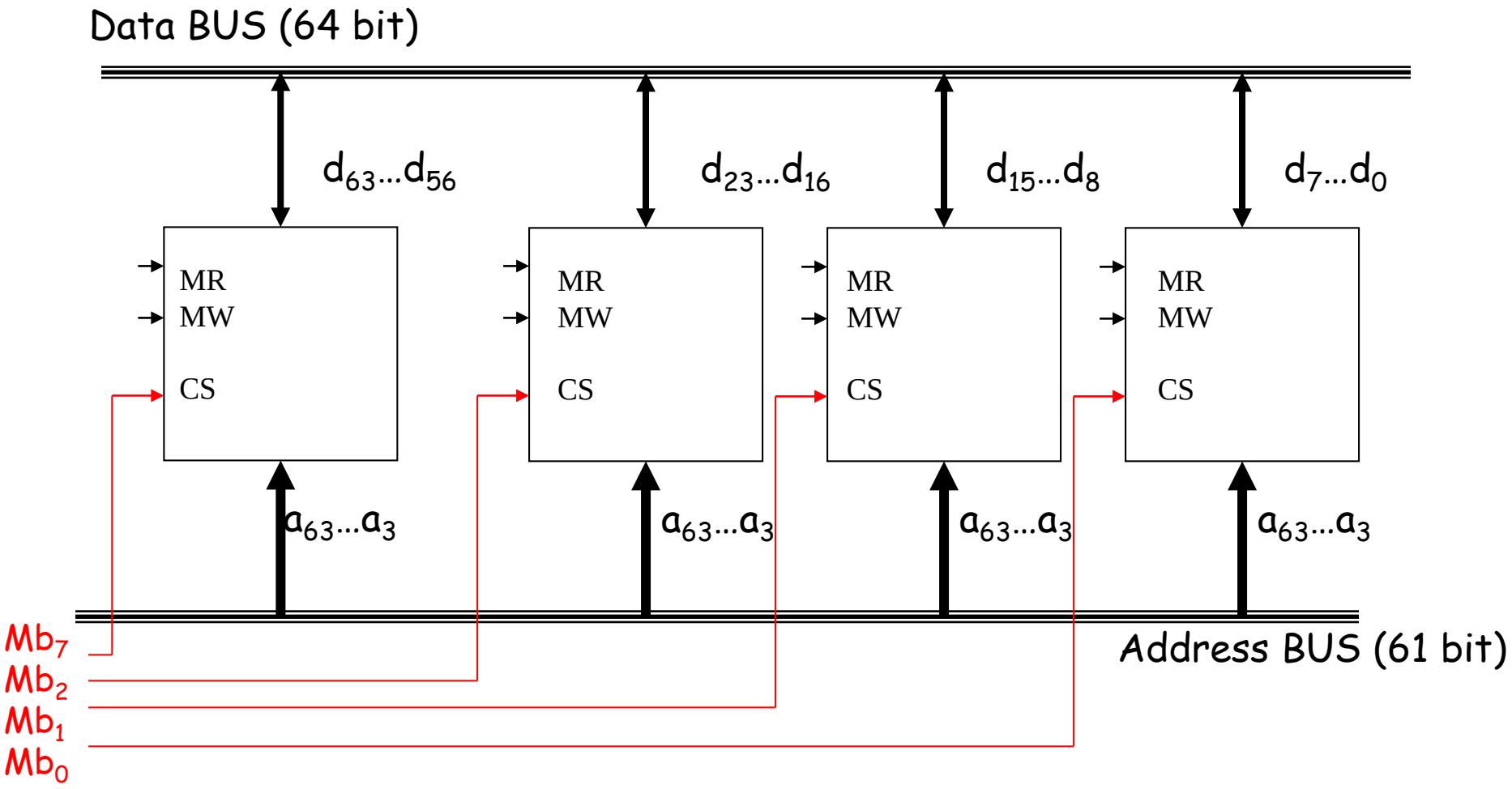


Esempio di memorizzazione di una informazione di otto byte allineati sullo stesso indirizzo di riga.



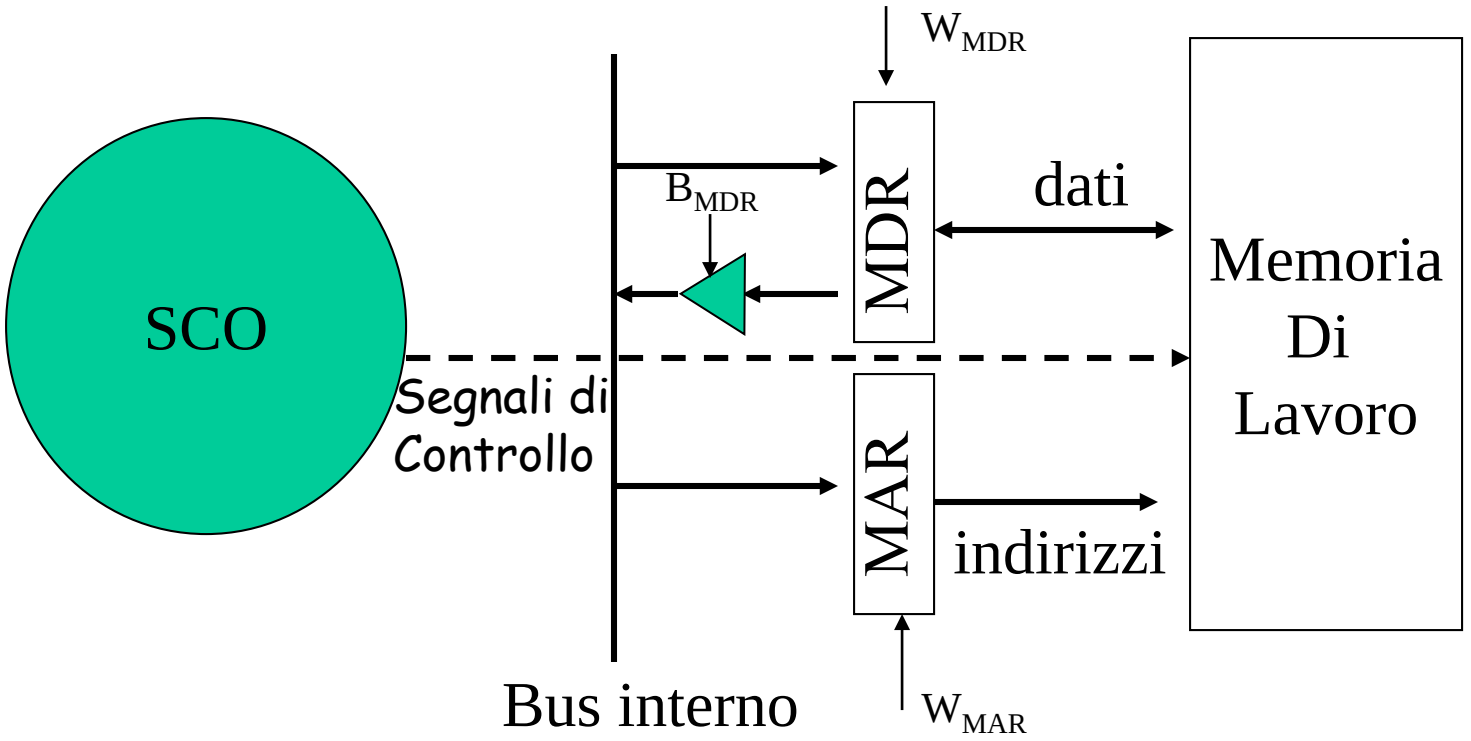
Esempio di memorizzazione di una informazione di due byte disallineati.

Hp: memoria con byte allineati

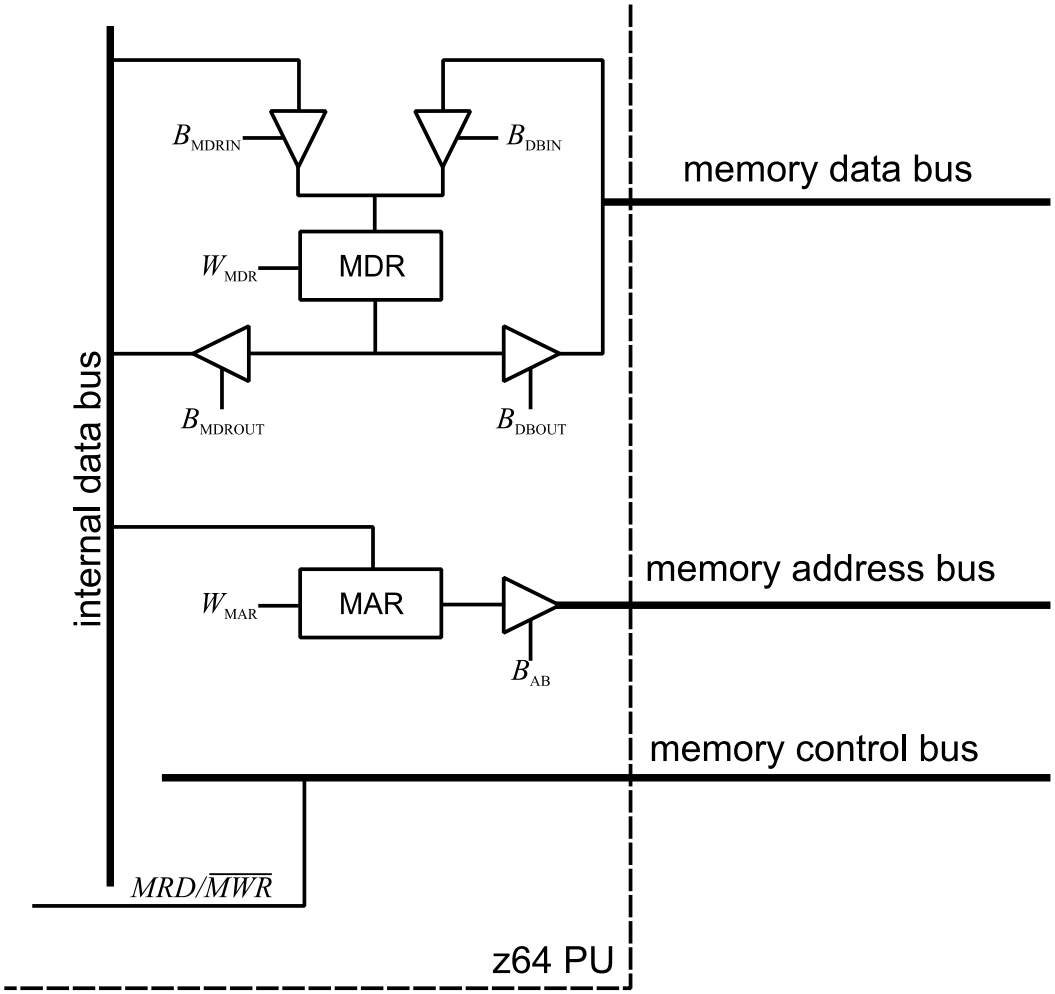


Memoria: interfaccia dello z64

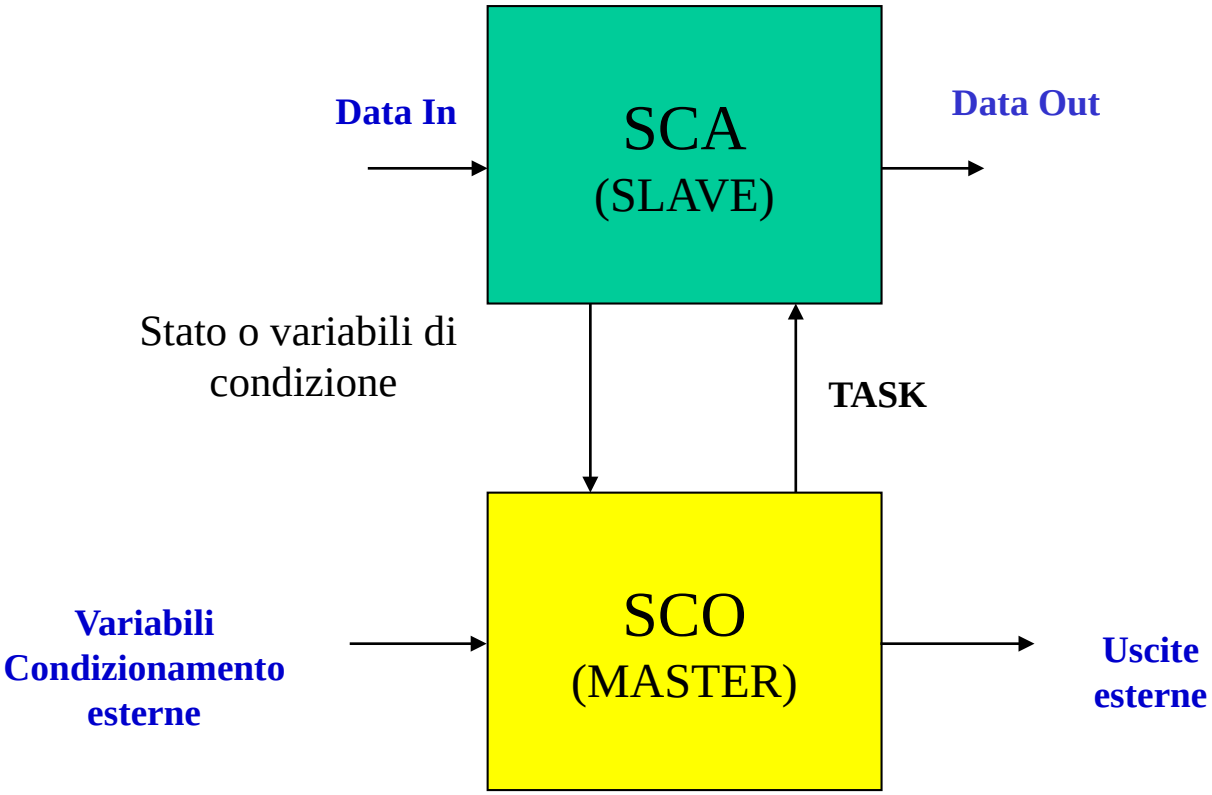
- Registro Memoria Dati (MDR)
- Registro Indirizzo (MAR)
- Segnali di Controllo (MR, MW, Mb_7, Mb_6, Mb_5, Mb_4, Mb_3, Mb_2, Mb_1, Mb_0)



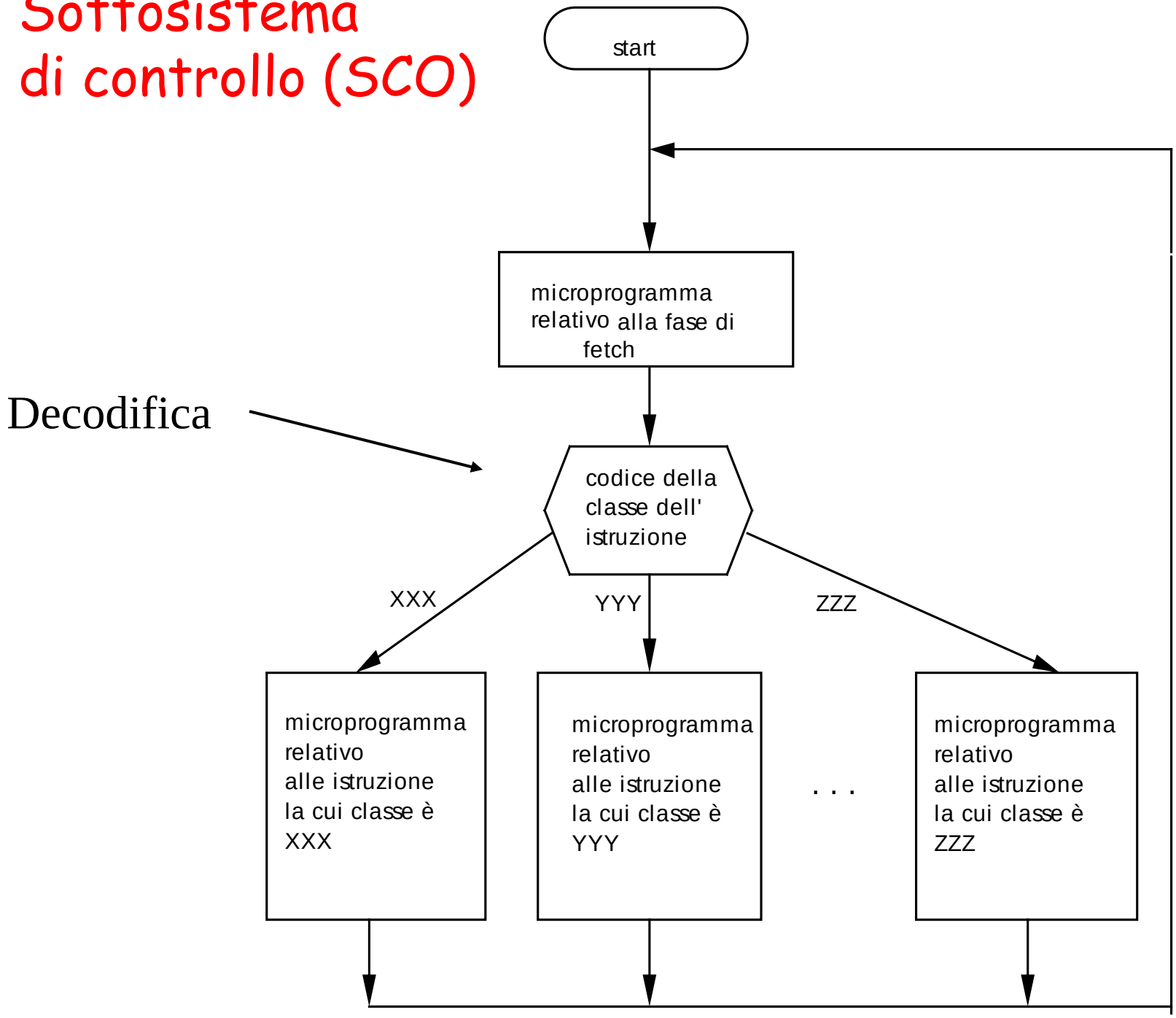
Memoria: interfaccia dello z64



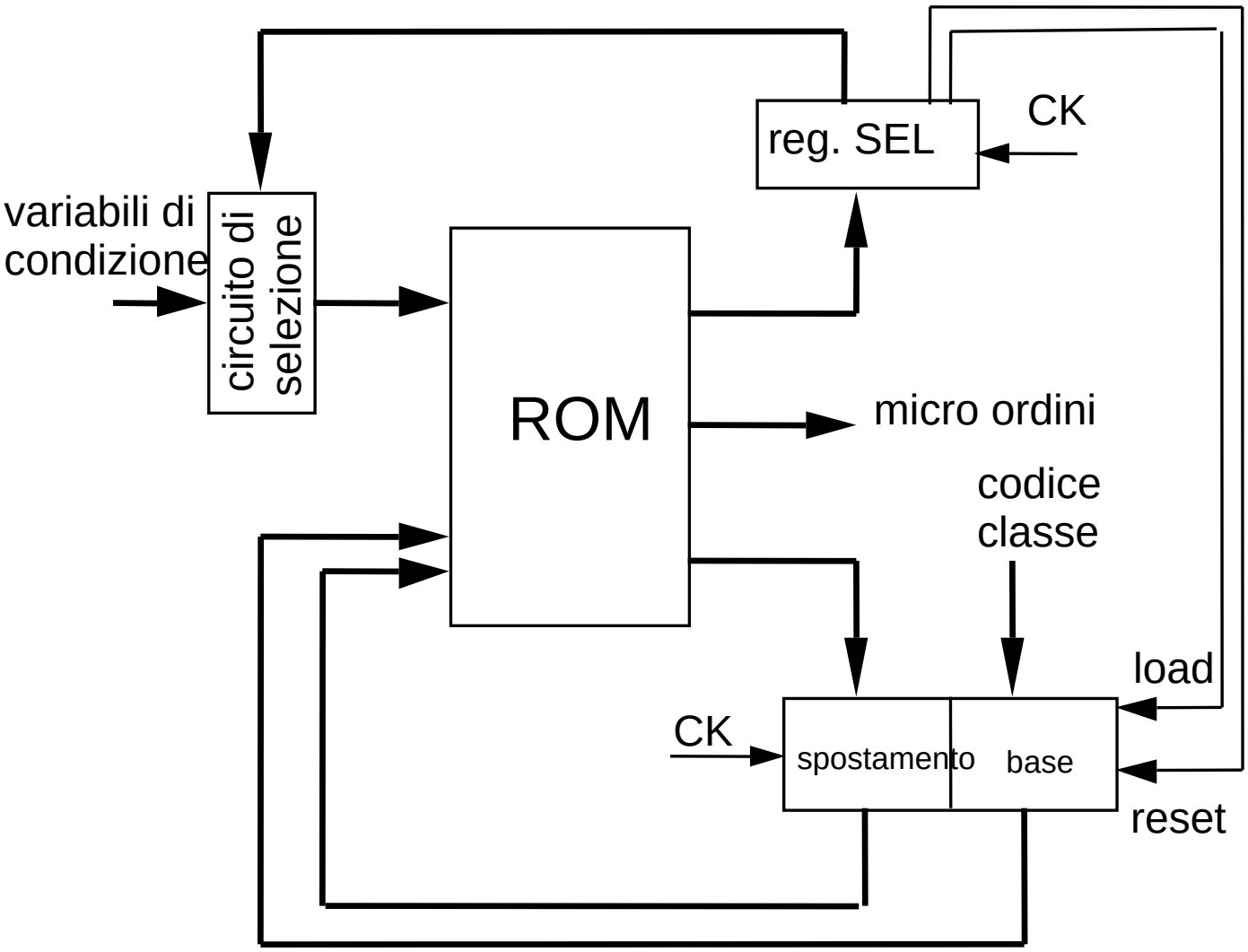
Sottosistema di controllo (SCO)



Sottosistema di controllo (SCO)



SCO: schema di Mealy



SCO: schema di Moore

Cod. Classe: codice istruzione

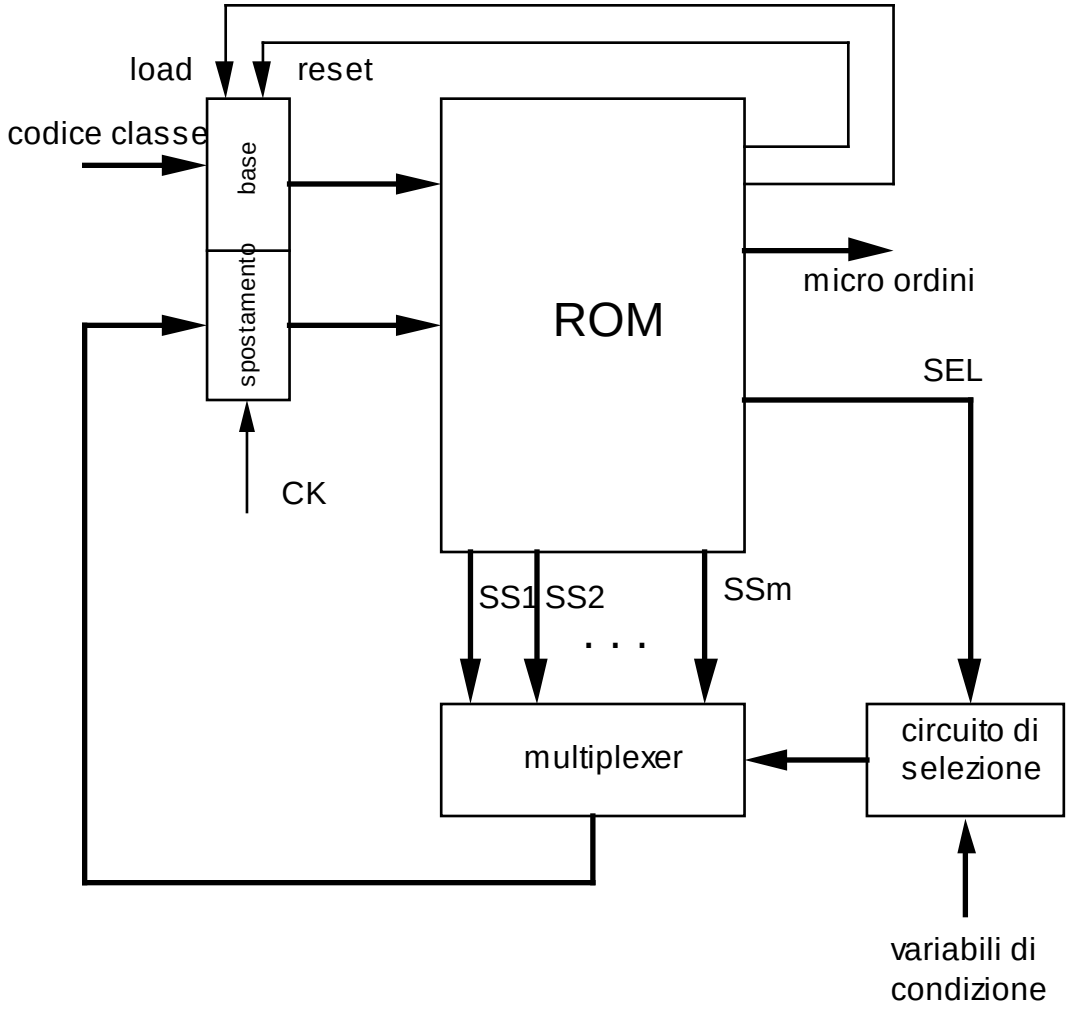
Load: segnale di caricamento nuova istruzione (a fine fetch)

Reset: azzera cod. classe (inizia fetch)

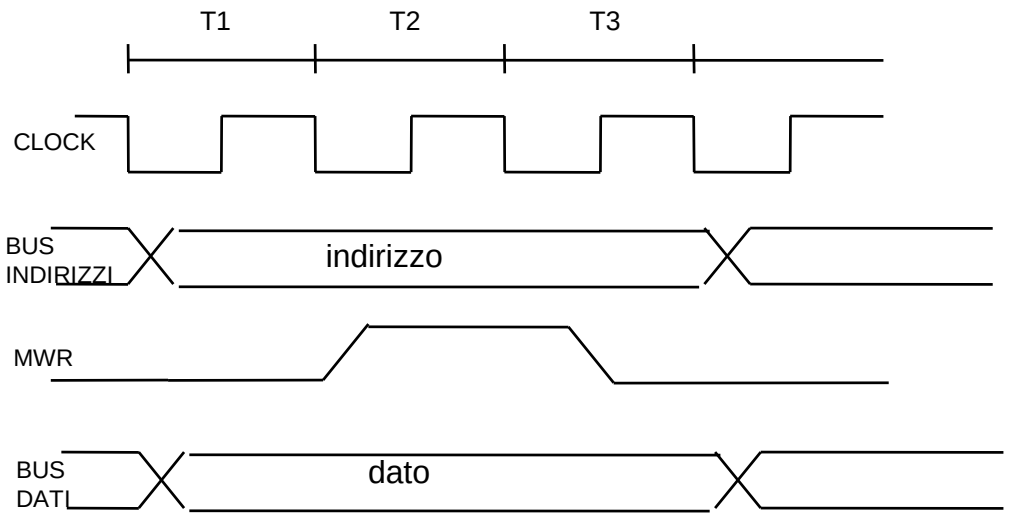
variab. di cond.: influenzano esecuzione istr. (ad es. var. in SR)

Microrordini: comandi per SCA

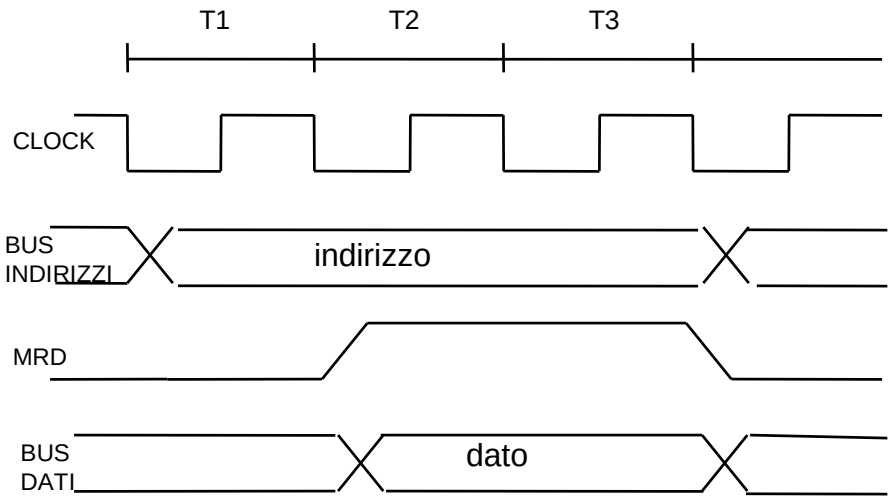
CK: clock



Interazione con la memoria: ciclo di scrittura

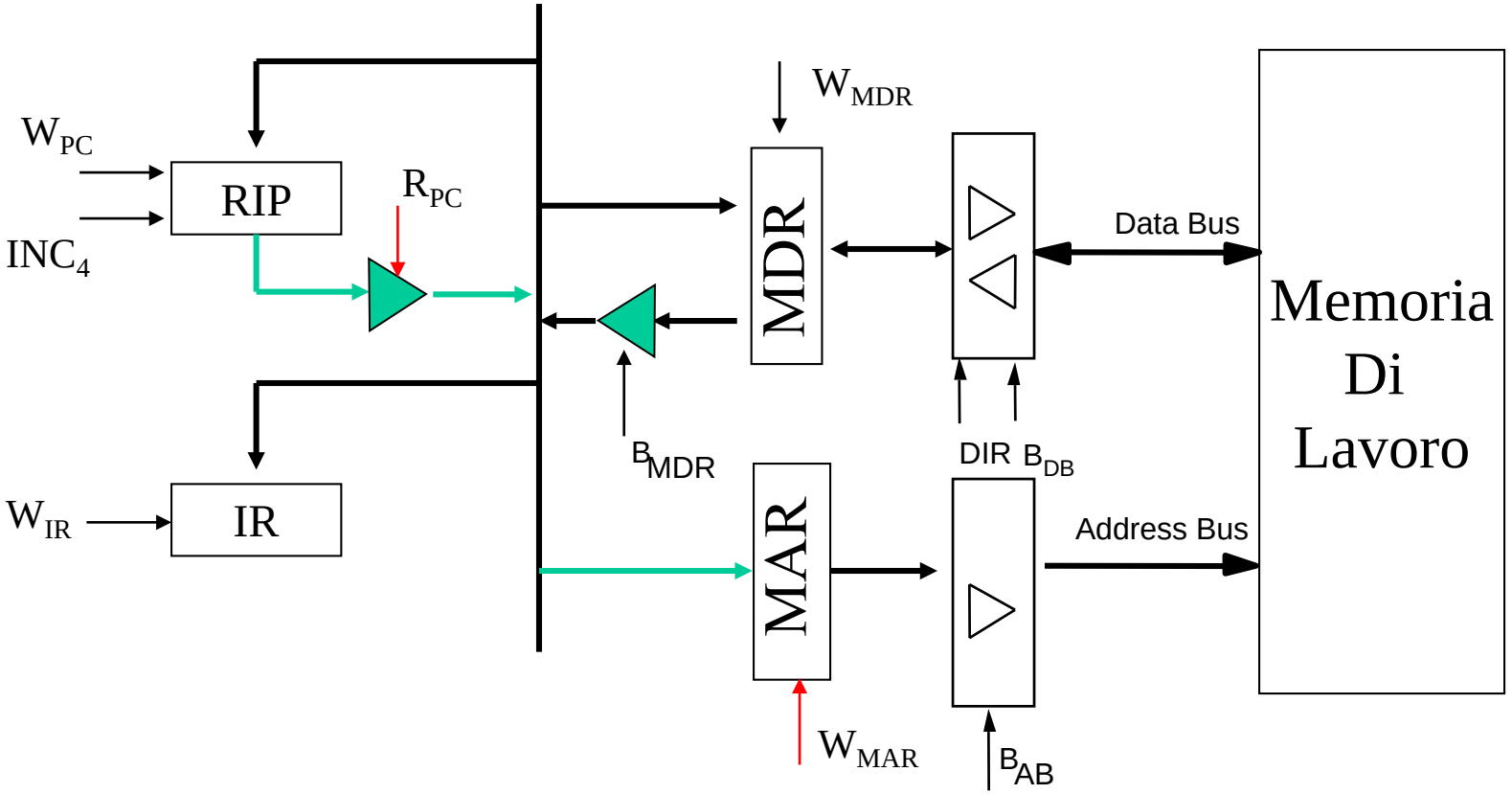


Interazione con la memoria: ciclo di lettura



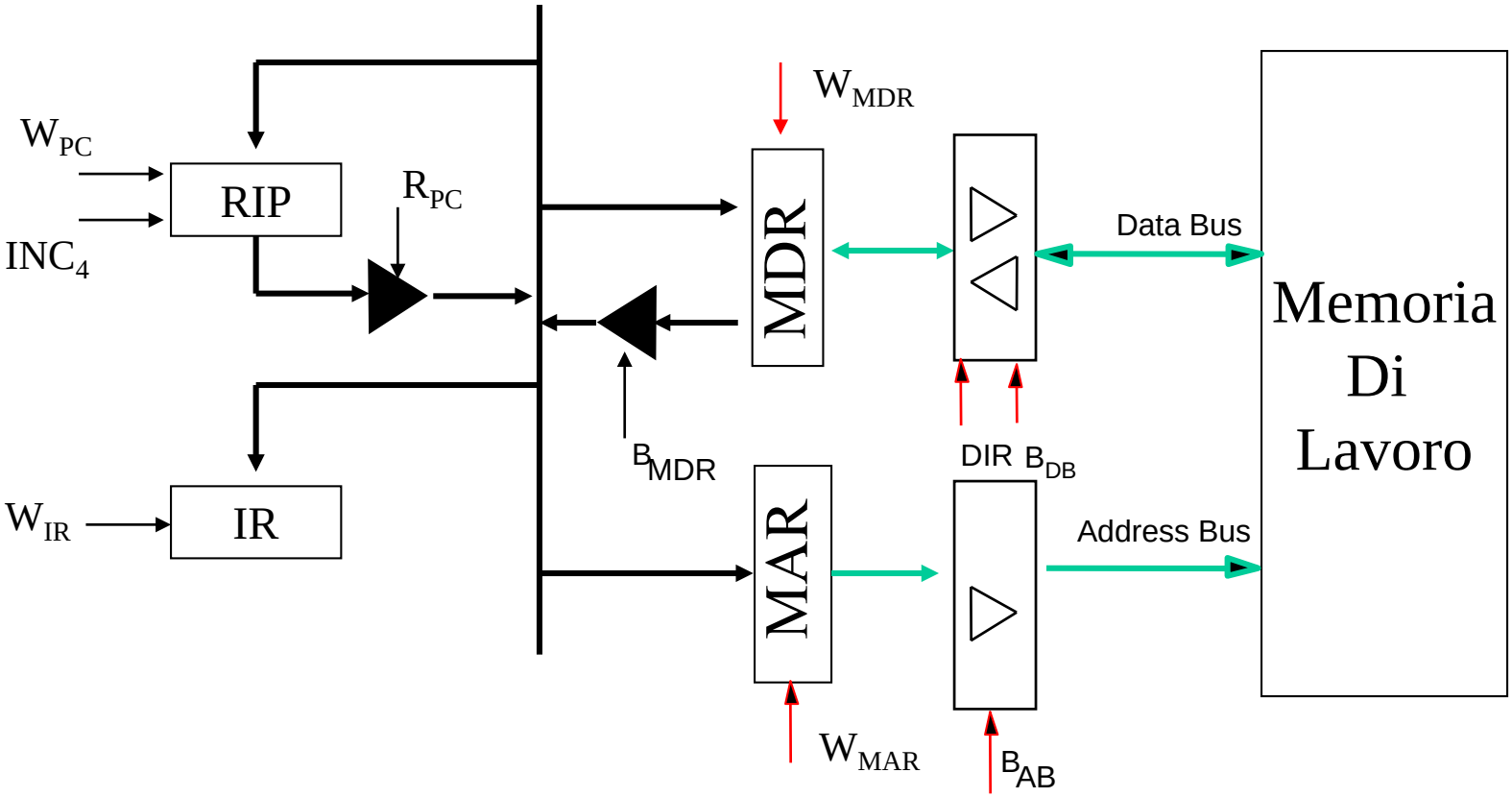
Passi elementari per eseguire il Fetch
(ipotesi: 8 byte allineati in memoria)

1. RIP -> MAR; /* trasferimento del contenuto del RIPnel MAR */



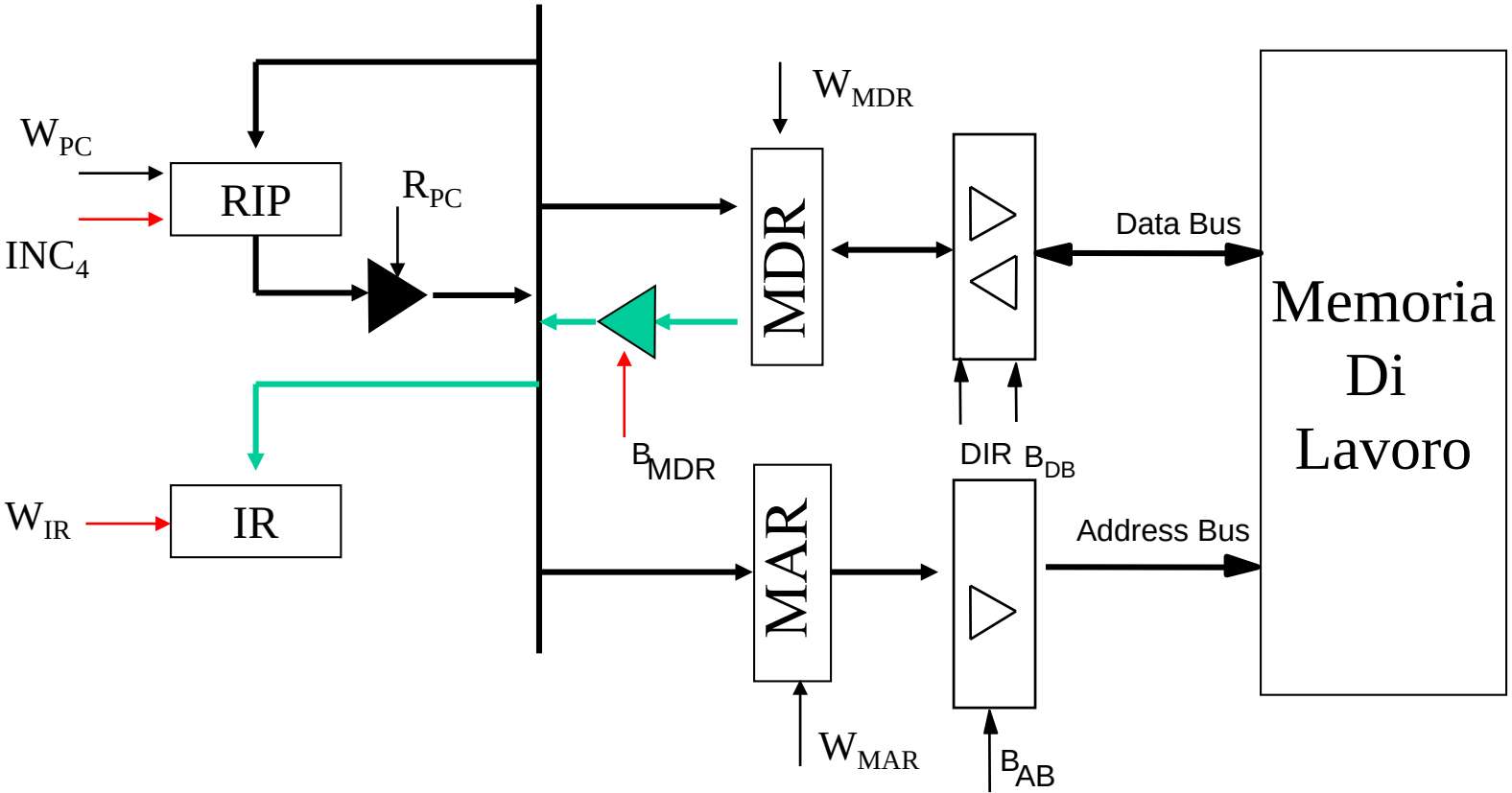
Fetch

1. RIP -> MAR; /* trasferimento del contenuto del RIP nel MAR */
2. (MAR)->MDR /* trasferimento istruzione da eseguire in MDR*/



Fetch

1. RIP -> MAR; /* trasferimento del contenuto del RIP nel MAR */
2. (MAR) -> MDR /* trasferimento istruzione da eseguire in MDR*/
3. MDR -> IR /* trasferimento istruzione da eseguire nell'IR*/
   RIP+8-> RIP /*e incr. RIP per prelievo prossima istruzione*/

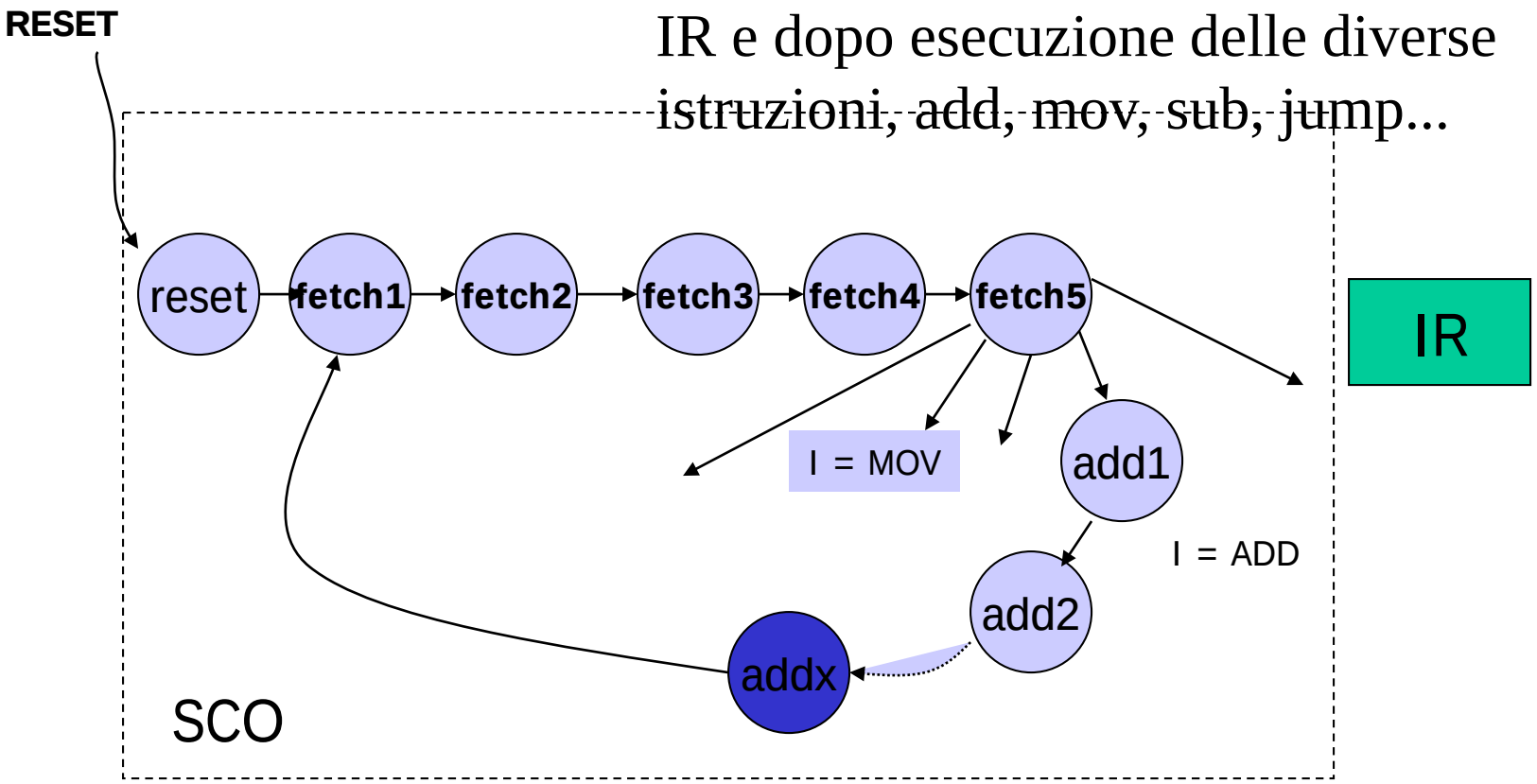


Fetch: micro-ordini

1. RIP -> MAR; /* trasferimento del contenuto del PC sul MAR */
    1. $R_{RIP} = 1, W_{MAR} = 1$
2. (MAR) -> MDR /* trasferimento istruzione da eseguire in MDR*/
    1. $B_{AB} = 1$ /* T1 */
    2. $B_{AB} = 1, MRD = 1$ /* T2 */
    3. $B_{AB} = 1, MRD = 1, W_{MDR} = 1$ /* T3*/
3. MDR -> IR /* trasferimento istruzione da eseguire in IR e predisposizione PC per prelievo prossima istruzione*/
    1. $B_{MDR} = 1, W_{IR} = 1, INC_8 = 1$

Ciclo Istruzione - Decode

fetch5: decodifica istr. utilizzando IR e dopo esecuzione delle diverse istruzioni, add, mov, sub, jump...



Esempio di esecuzione di istruzioni

Nello z64 la fase di esecuzione di un ciclo istruzione consiste in un numero variabile di cicli macchina dipendente dal numero di accessi in memoria necessari (oltre al fetch)

ADDQ R8, R9

Entrambi gli operandi sono contenuti in registri interni del Z64 (indirizzamento a registro)

1. RIP -> MAR;
2. (MAR) -> MDR
3. MDR -> IR , RIP+8->RIP
4. R8 -> Temp1
5. R9 -> Temp2
6. OUT_ALU -> R9

ADDQ #20h, R9

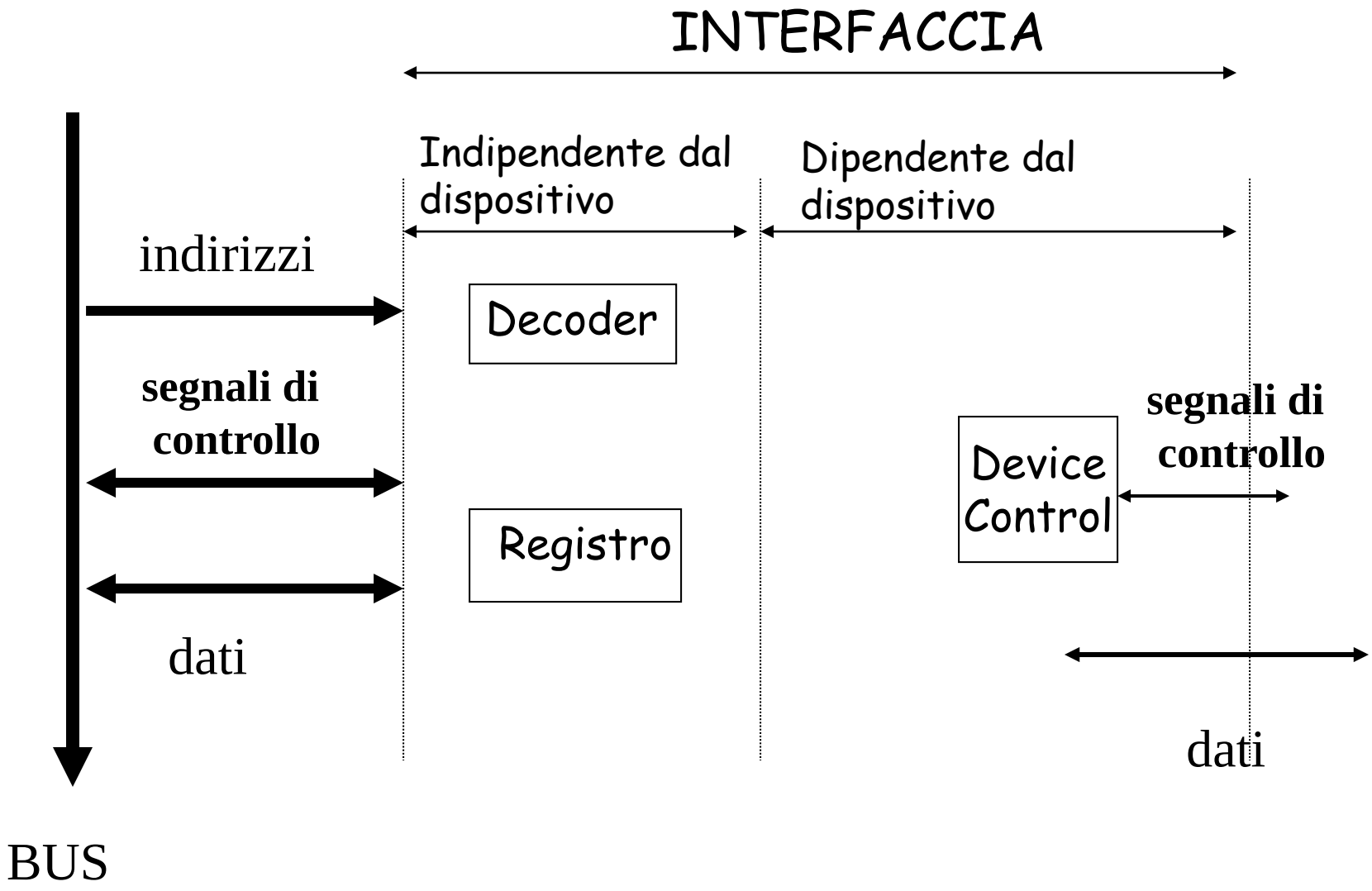
Uno degli operandi (0x20) è memorizzato nei due byte successivi a quelli contenente l'istruzione (indirizzamento immediato)

1. RIP -> MAR;
2. (MAR) -> MDR
3. MDR -> IR , RIP+8->RIP
4. R9 -> Temp1
5. RIP -> MAR
6. (MAR) ->MDR
7. MDR -> Temp2, RIP+8->RIP
8. OUT_ALU -> R9

z64 - Interazione con l'esterno

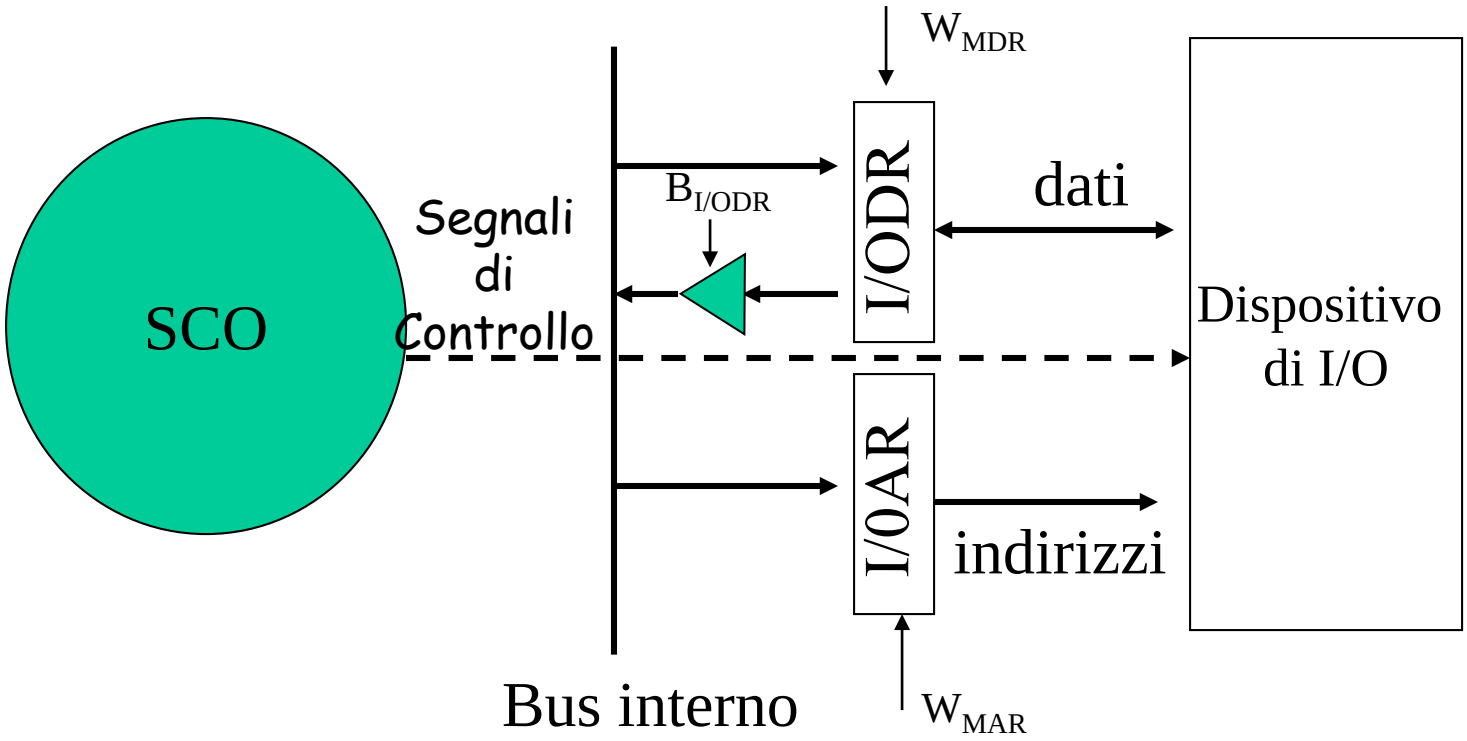
- Ogni fase che comporta l'interazione con le unità esterne viene detta ***ciclo macchina***.
- Ogni ciclo macchina può essere costituito da uno o due ***cicli di bus***; per esempio la lettura di una parola memorizzata su due byte non allineati sullo stesso indirizzo di riga necessita di due accessi in memoria (cioè di due cicli di bus).

Interfaccia dispositivi di I/O

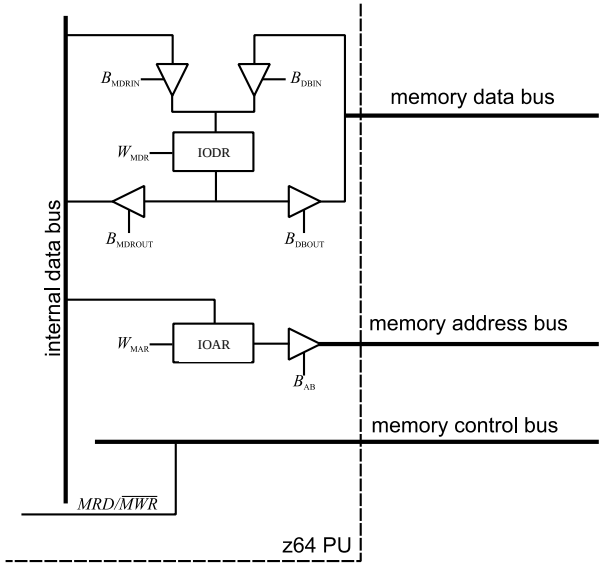


Dispositivi di I/O: interfaccia dello z64

- **Registro Dati (I/ODR)**
- **Registro Indirizzo (I/OAR)**
- **Segnali di Controllo (I/OR,I/OW, Start, …….)**



I/O: Interfaccia dello z64



Interconnesione dello z64 con due bus