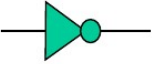
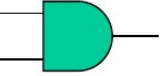
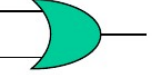

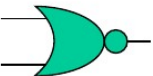
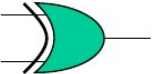
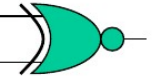


	Operatore	Simbolo	Proprietà
	NOT	$y = \neg x$	$y=1$ se e solo se $x=0$
	AND	$y = x_1 x_2$	$y=1$ se e solo se $x_1=x_2=1$
	OR	$y = x_1 + x_2$	$y=0$ se e solo se $x_1=x_2=0$
	NAND	$y = x_1 / x_2$	$y=0$ se e solo se $x_1=x_2=1$
	NOR	$y = x_1 \downarrow x_2$	$y=1$ se e solo se $x_1=x_2=0$
	XOR	$y = x_1 \oplus x_2$	$y=1$ se e solo se $x_1 \neq x_2$
	XNOR	$y = x_1 \equiv x_2$	$y=1$ se e solo se $x_1=x_2$

Inverter tree state NON è una porta logica

L'uscita può assumere uno stato di alta impedenza elettrica, utile per disconnettere l'uscita dagli altri circuiti ad essa collegati

Una **rete logica** è un circuito elettronico digitale in grado di realizzare una o più funzioni di commutazione

Il **codificatore** realizza la funzione di codifica binaria, ossia associare ad ogni elemento di un insieme T composto da m simboli, una sequenza distinta di n bit

Il **decodificatore** realizza la funzione inversa del codificatore, a partire da una parola di un codice binario genera una uscita che identifica uno dei simboli dell'insieme T

Per ogni configurazione di ingresso, una sola uscita vale 1, le altre hanno valore 0

Multiplexer ha $m = 2^n$ ingressi e in uscita una fra le m a seconda del controllo

Una macchina sequenziale è una quintupla $MS = (I, S, O, \delta, \omega)$

- I = alfabeto degli stati
- S = insieme degli stati
- O = alfabeto di uscita
- δ = funzione dello stato successivo
- ω = Funzione di uscita (di tipo mealy o di tipo moore)

Tabelle degli stati/uscite

- **MACCHINA DI MEALY**

Matrice $|S|$ righe per $|I|$ colonne.

L'elemento in posizione h,k contiene il prossimo stato e l'uscita nel caso in cui lo stato corrente sia h e l'ingresso sia il k -esimo

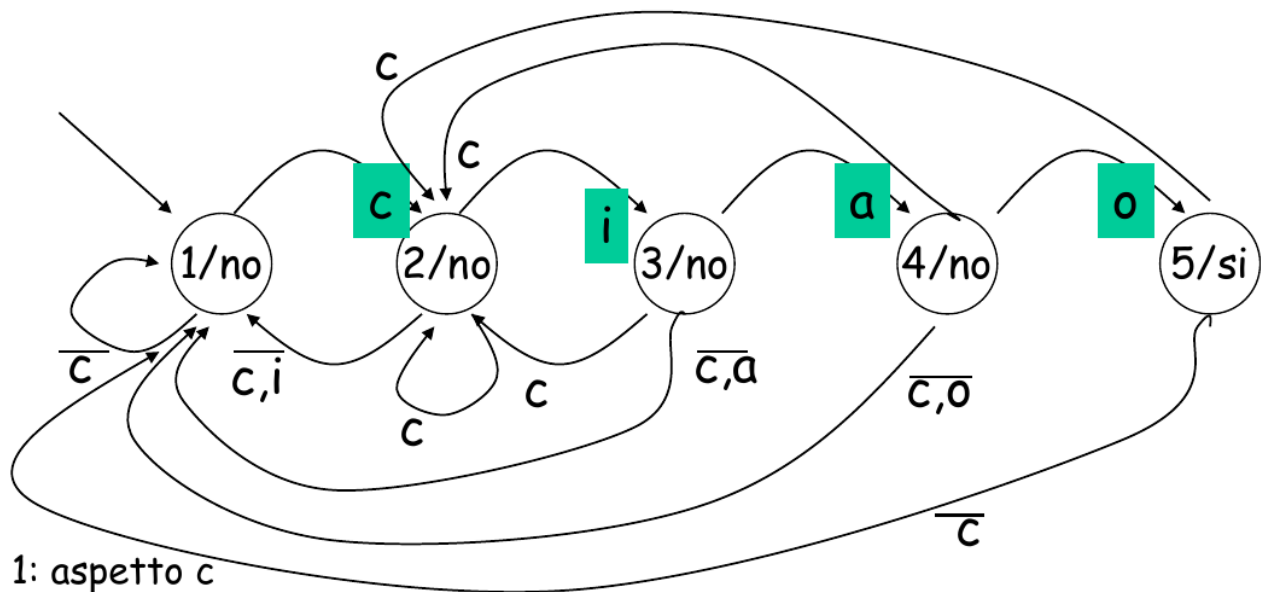
- **MACCHINA DI MOORE**

Matrice $|S| \times |I| + 1$.

L'elemento in posizione h,k contiene il prossimo stato nel caso in cui lo stato corrente sia h e l'ingresso sia il k -esimo

L'elemento $h, |I| + 1$ contiene l'uscita nel caso in cui lo stato sia h

Diagramma degli stati (Moore)



1: aspetto c

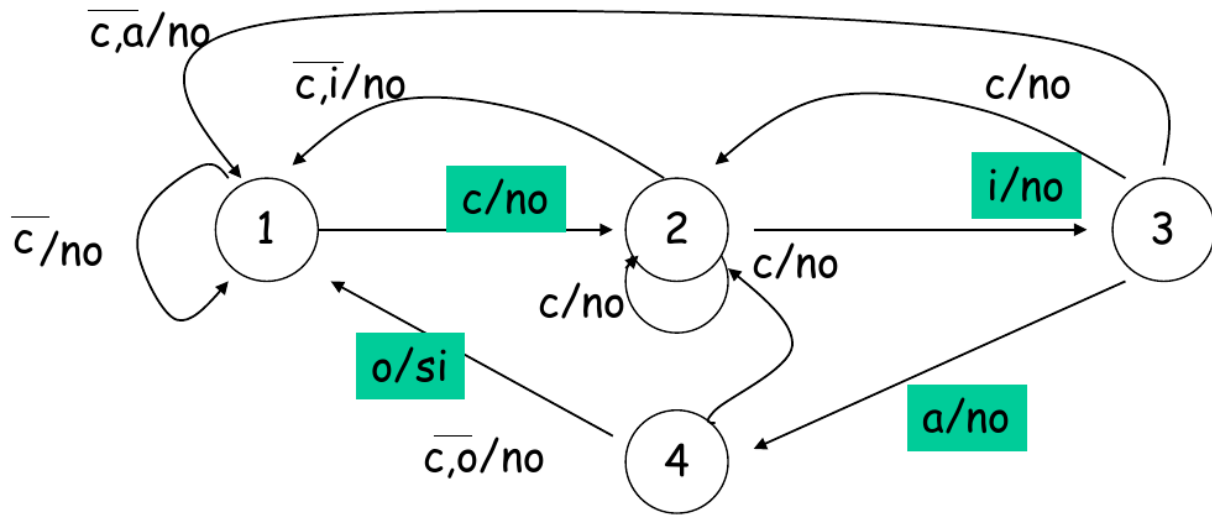
2: aspetto i

3: aspetto a

4: aspetto o

5: parola completa

Diagramma degli stati (Mealy)



- 1: attesa c
- 2: attesa i
- 3: attesa a
- 4: attesa o

Flip-flop S-R (Set-Reset) o bistabile (macchina asincrona)

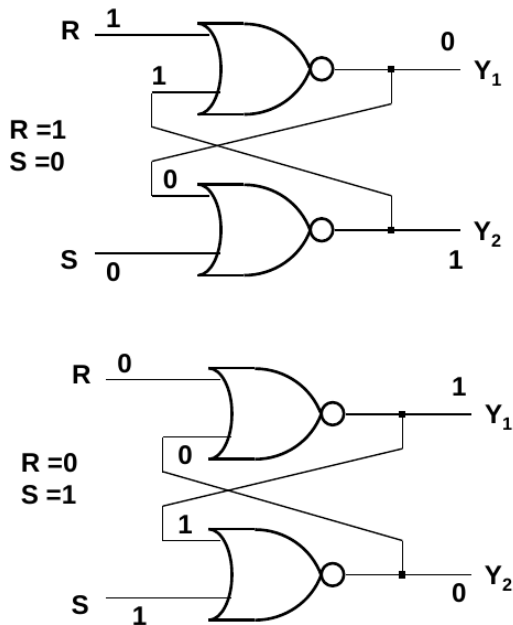
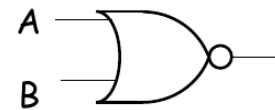


Tabella verità NOR

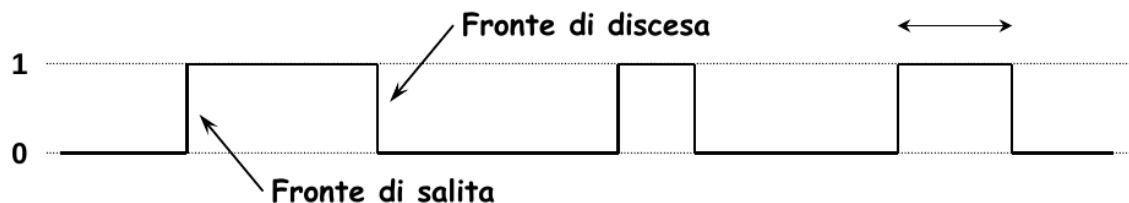
A \ B	0	1
0	1	0
1	0	0



- Se un ingresso è uguale ad 1 allora l'uscita vale 0
- Se un ingresso è uguale a 0 allora l'uscita è uguale al valore dell'altro ingresso negato

Segnale di sincronizzazione

- Un segnale di sincronizzazione è una variabile binaria che viene utilizzata per abilitare la commutazione di un flip-flop (sincronizzato)
- L'abilitazione alla commutazione può essere fatta:
 - all'istante in cui avviene la commutazione della variabile da 0 ad 1 (**fronte di salita**);
 - All'istante in cui avviene la commutazione della variabile da 1 a 0 (**fronte di discesa**)
 - Nel periodo in cui è stabile ad 1 oppure a 0 (a **livello**)



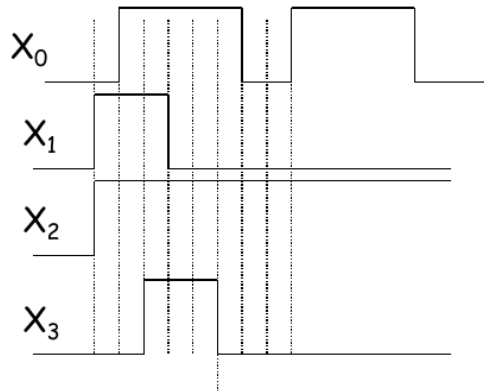
- nella realtà le transizioni 0→1 e 1→0 non sono istantanee

Segnale di sincronizzazione (cont.)

- Alcune volte il **segnale di abilitazione** per la commutazione può avere un comportamento periodico (periodo T), in questi casi viene chiamato anche **clock (CK)**
- Spesso il segnale di abilitazione per la commutazione viene identificato con CK anche se non ha un comportamento periodico



Classificazione variabili di ingresso



X_0 a livello rispetto a X_1
 X_1 a livello rispetto a X_0

X_0 impulsiva rispetto a X_2
 X_2 a livello rispetto a X_0

X_0 a livello rispetto a X_3
 X_3 impulsiva rispetto a X_0

Si cerca di evitare il comportamento
come quello presente tra X_1 e X_2 che
commutano "contemporaneamente"

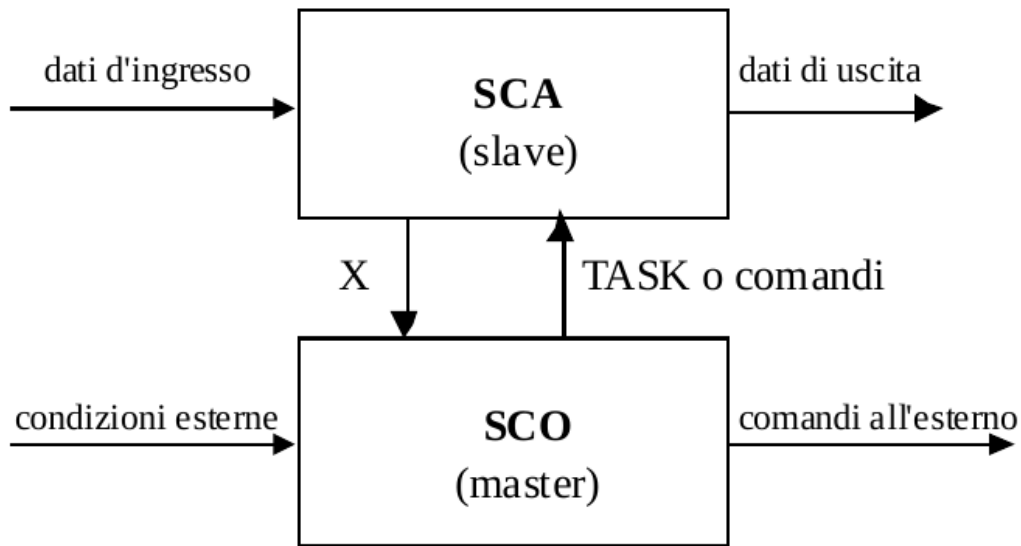
(possibilità di alee-corse, si studiano a Reti Logiche)
Le alee si verificano spesso se i segnali
vengono generati da fenomeni naturali (non
controllabili dall'uomo), p.e. nei contatori
Geiger, interferenze....

Reti LLC (LEVEL LEVEL CLOCKED)

La rete sequenziale lavora con le seguenti ipotesi:

- Variabili d'ingresso di tipo a livello (ossia i valori in ingresso rimangono fissi per un periodo T sufficientemente lungo per far assumere all'uscita il nuovo valore di regime, ossia $T > d$)
- Variabili di uscita a livello
- Segnale di abilitazione "positive or negative edge trigger", o a livello (in quest'ultimo caso la variabile di commutazione deve essere pari a 1 per un periodo di tempo sufficiente per far commutare i flip-flop, ma inferiore al minimo tempo di commutazione dei circuiti combinatori che calcolano lo stato successivo, altrimenti si potrebbero avere più computazioni)

SISTEMI DIGITALI COMPLESSI

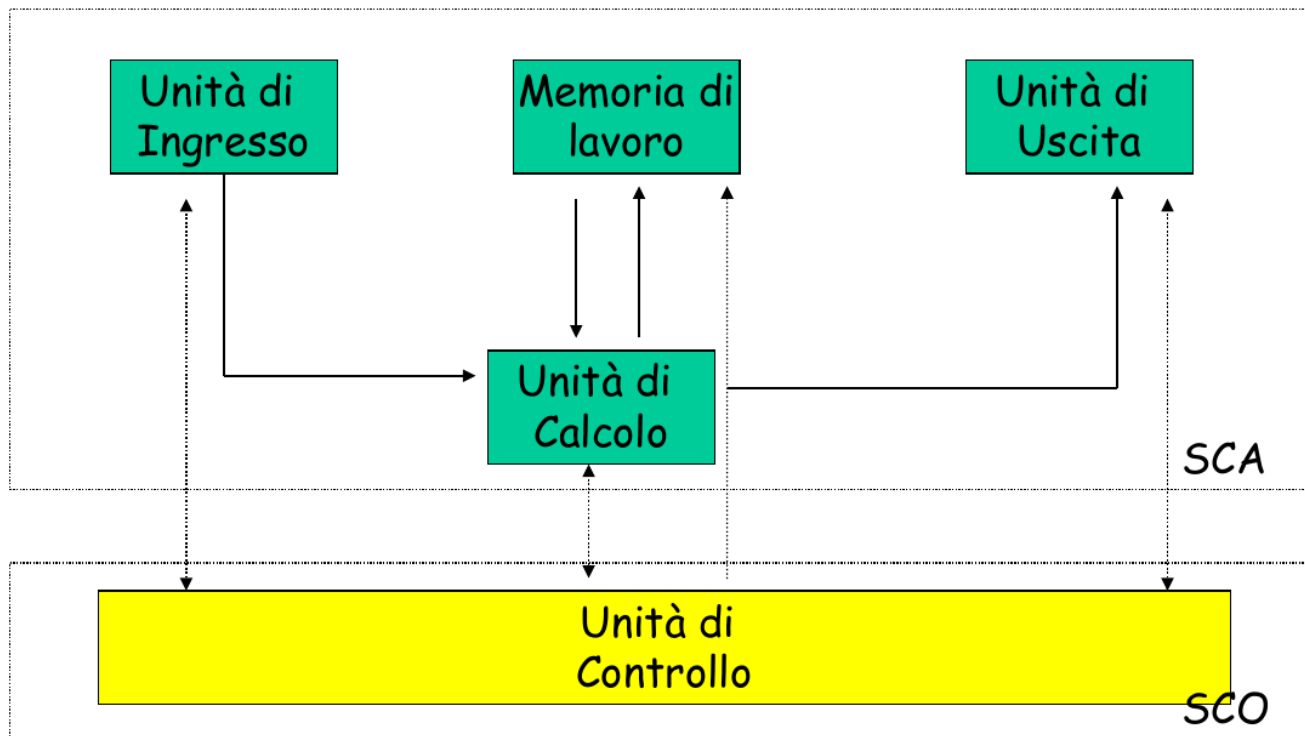


Sistema digitale complesso suddiviso in SCO-SCA

Procedimento di sintesi di un sistema digitale, può essere suddiviso nei seguenti passi:

- 1) Specifica del problema
- 2) Individuazione di un algoritmo di soluzione
- 3) Progetto di un SCA atto a supportare l'algoritmo
- 4) Definizione di un SCO che implementa l'algoritmo
- 5) Valutazione del sistema: se le prestazioni rispondono alle specifiche del problema si passa al punto 6, altrimenti si verifica se è possibile definire un altro SCA; in caso positivo di modifica il SCA e si torna al punto 4; se no si passa al punto 2
- 6) Sintesi del sistema e verifica del corretto funzionamento

Suddivisione SCA-SCO



- > **Segnali di controllo/condizione**
—————> **Flusso dati**

Z64: Bus Interno

- Usato per il collegamento dei registri interni
- Operazioni che caratterizzano il bus:

1) Ricezione dati:

- I bit presenti sul bus sono memorizzati in un registro

2) 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:

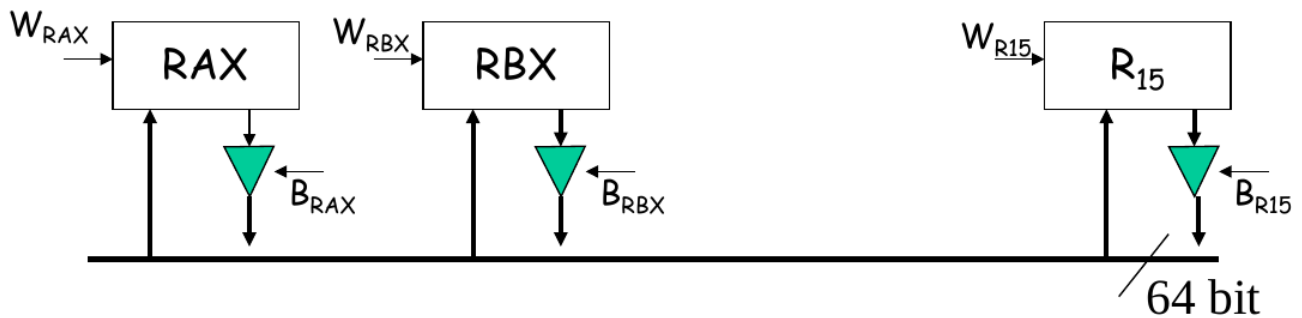
- Il segnale di abilitazione alla scrittura di un registro corrisponde alla ricezione dei dati presenti sul bus in quel momento
- Il segnale di abilitazione sul buffer three-state permette di trasferire sul bus il contenuto del registro

Z64 BUS INTERNO

z64: BUS interno, segnali di controllo

hp: operandi a 64 bit

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



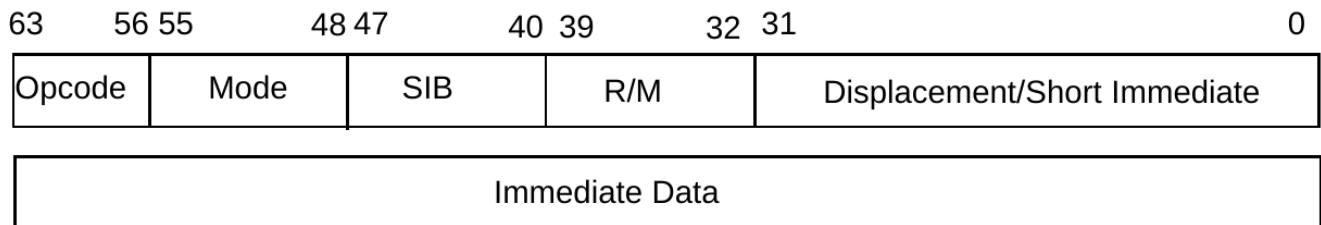
$W_i=1$, scrivi il contenuto del bus sul registro i
 $B_i=1$ scrivi sul bus il contenuto del registro i

Codifica 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

Formato istruzione

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

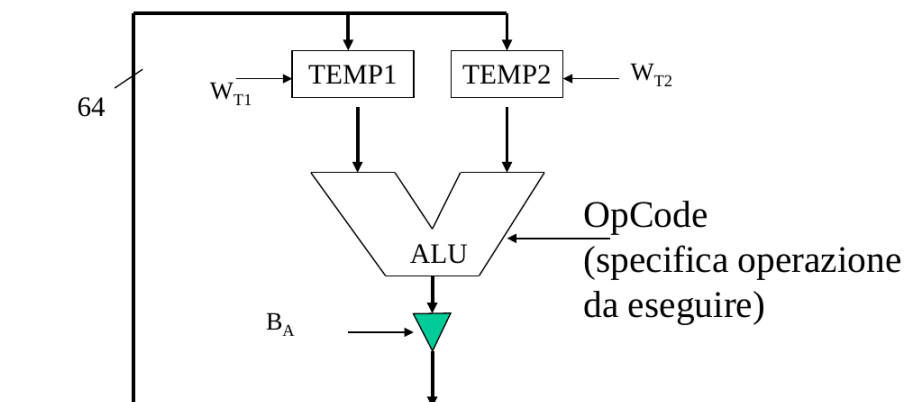
Displacement/Short Immediate: Indica se c'è un displacement nell'accesso in memoria
o il valore dell'immediato a 32 bit

Immediate Data: valore del dato immediato a 64 bit

ALU

z64- ALU

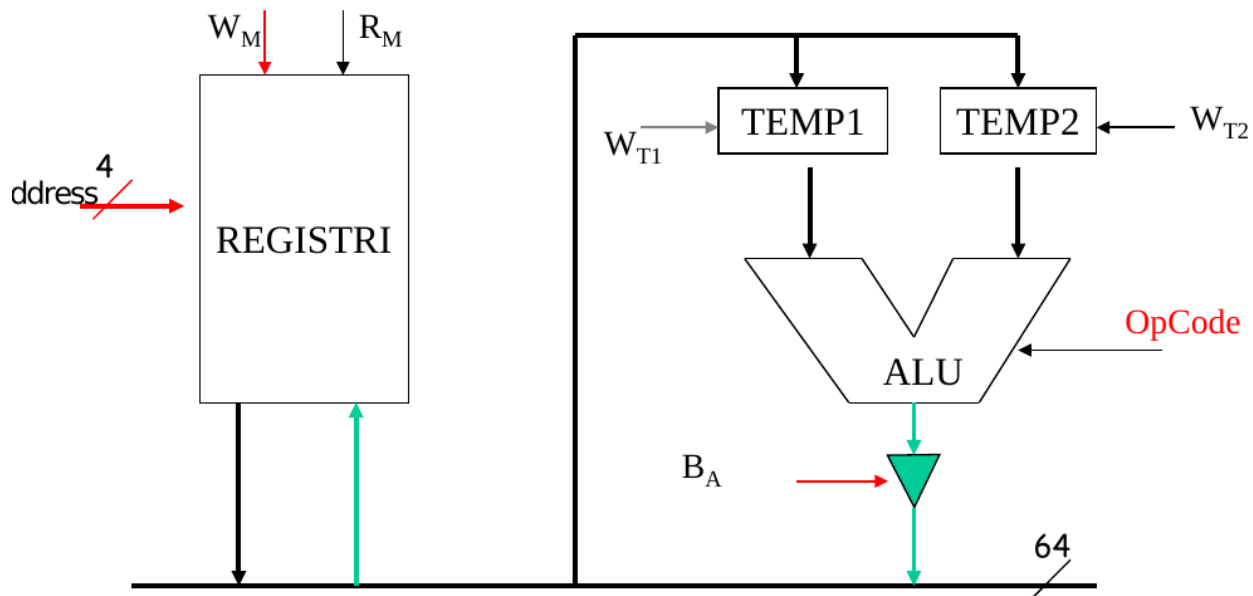
- Esegue le operazioni aritmetiche e logiche dei valori memorizzati in due registri tampone (non visibili al programmatore) Temp1 e Temp2
- Il risultato è posto in un registro generale Ri



ALU in esecuzione add

z64 - ALU, esempio: esecuzione addw R2,R1

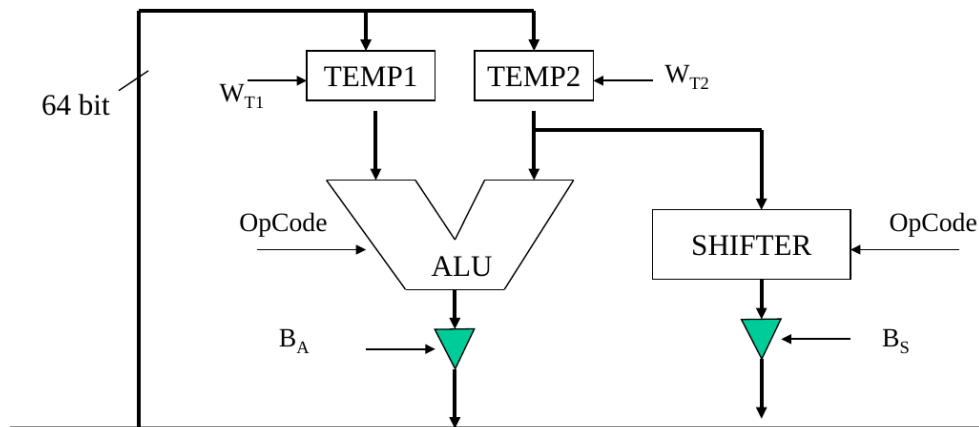
1. **R8 -> Temp1** $R_M=1$, Address = 1000, $W_{T1} = 1$
2. **R15 -> Temp2** $R_M=1$, Address = 1111, $W_{T2} = 1$
3. **ALU-OUT(Temp1+Temp2)->R15**
 $W_M=1$, Address = 1111, OpCode = addw, $B_A=1$



Z64-Shifter

z64 - Shifter

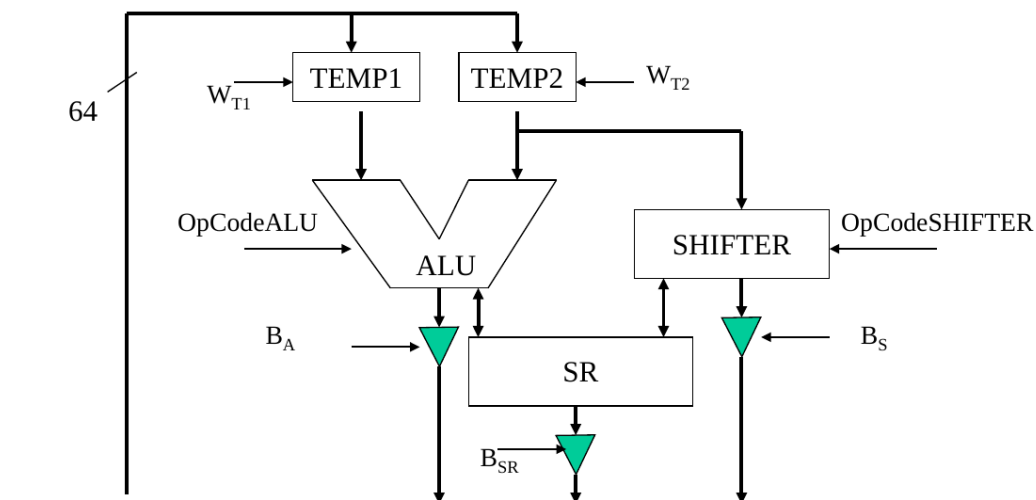
Usato per eseguire operazioni di scorrimento di posizioni, nonché per lo spostamento di dati tra registri interni (i registri tampone non possono scrivere sul bus mentre i segnali di controllo valgono per tutti i registri)



Status Register

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

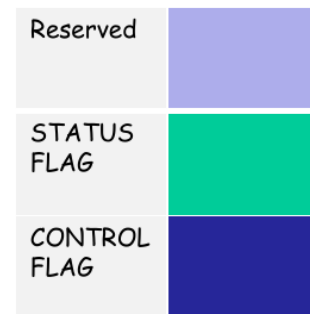
IF: Interrupt Flag

SF: Sign Flag

ZF: Zero Flag

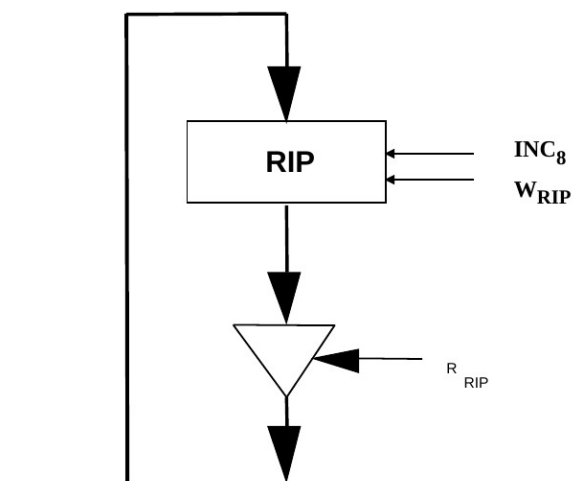
PF: Parity Flag

CF: Carry Flag



Incremento RIP

Incremento RIP

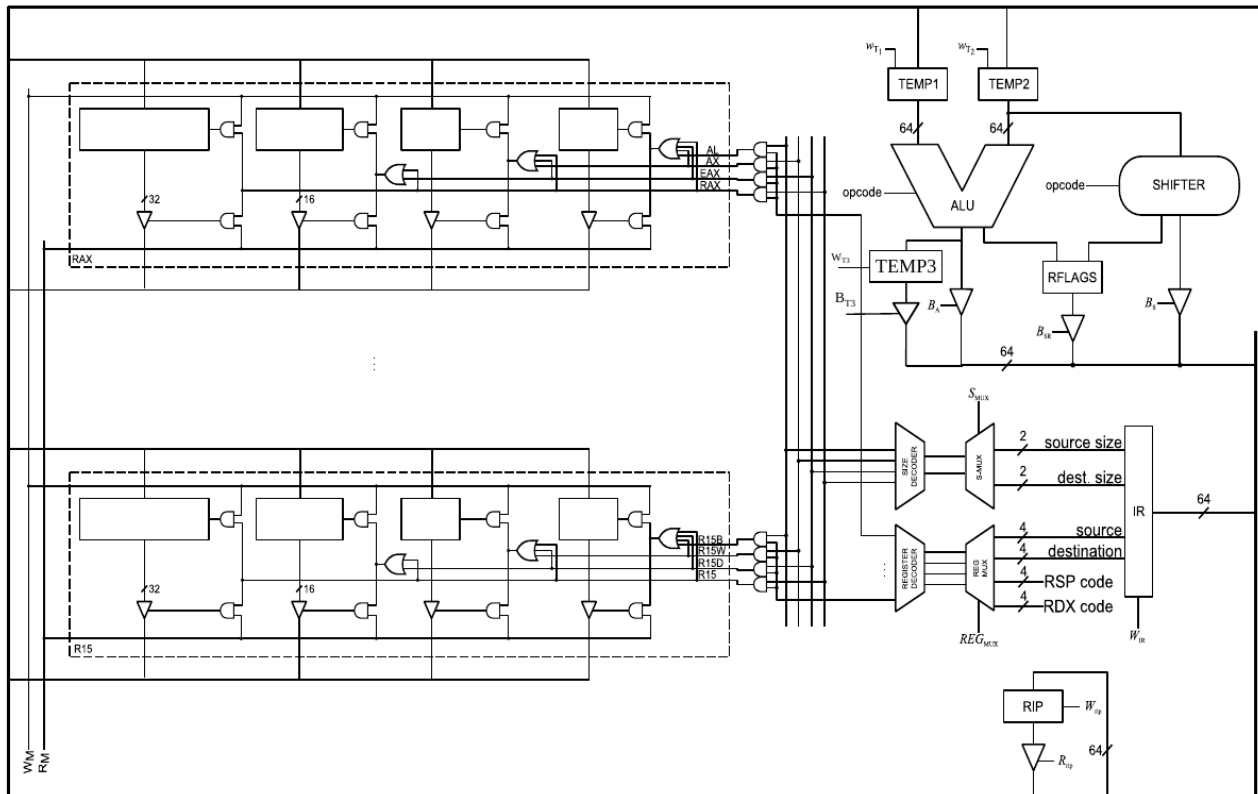


Il RIP deve essere incrementato (se non si eseguono istruzione di salto)

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

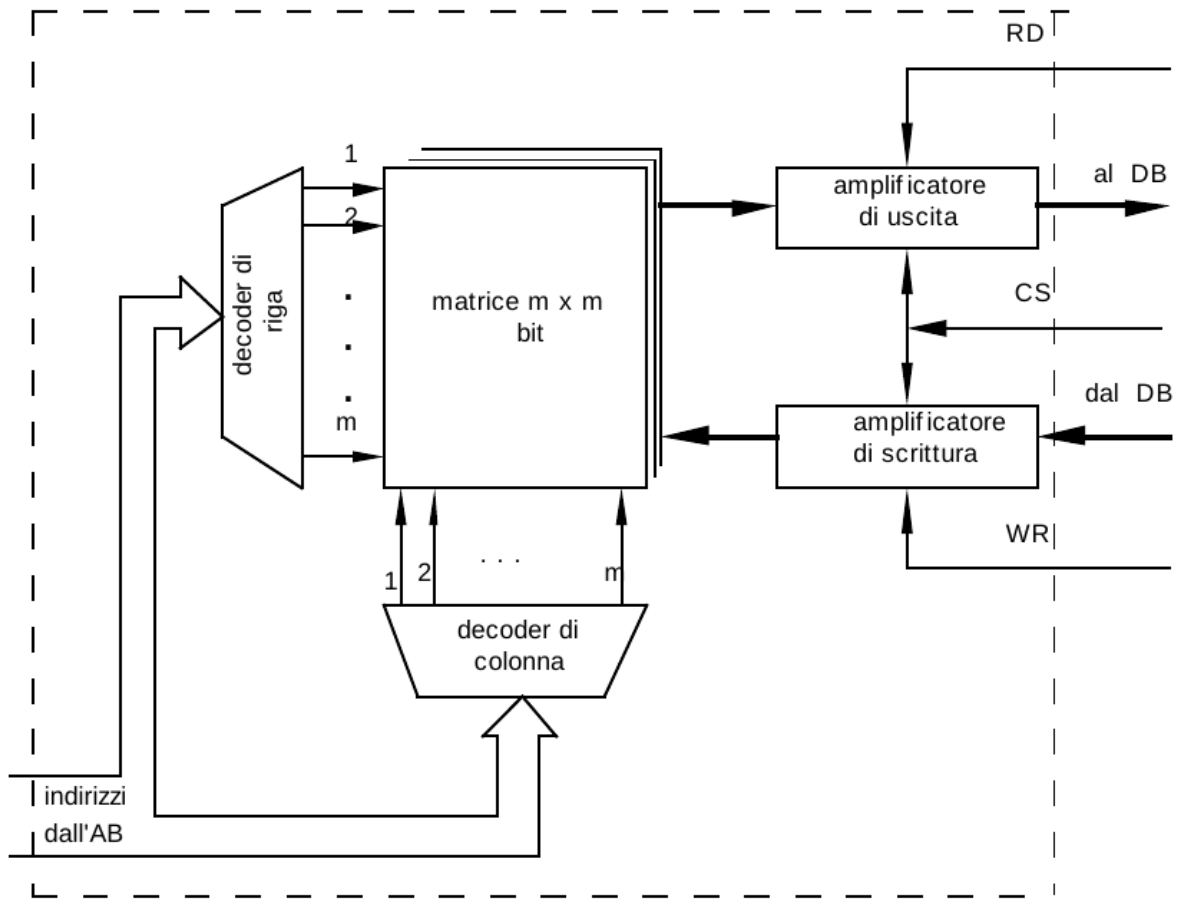
Architettura senza interfacce

Architettura senza interfacce



Memoria Ram statica

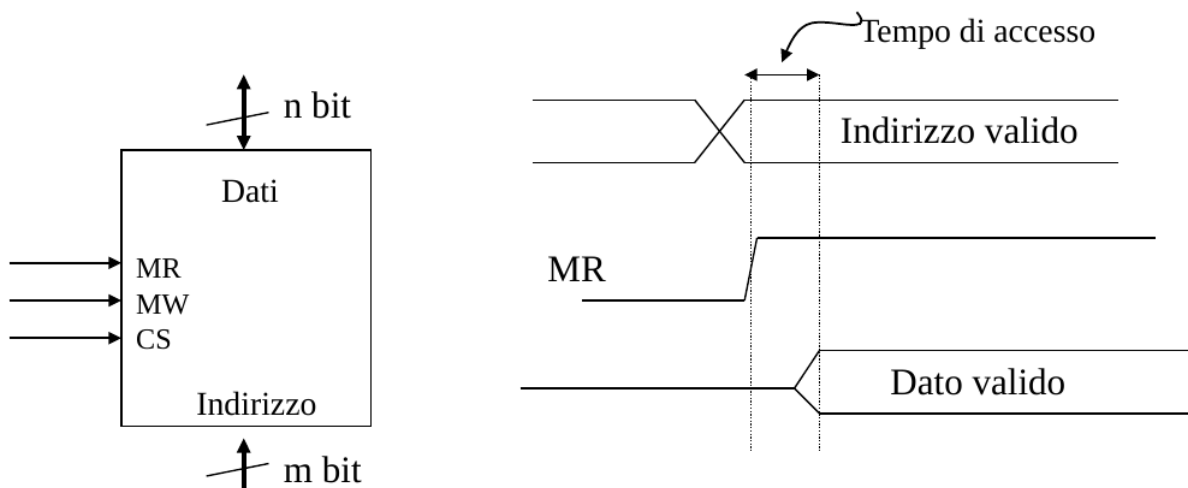
Memoria RAM statica



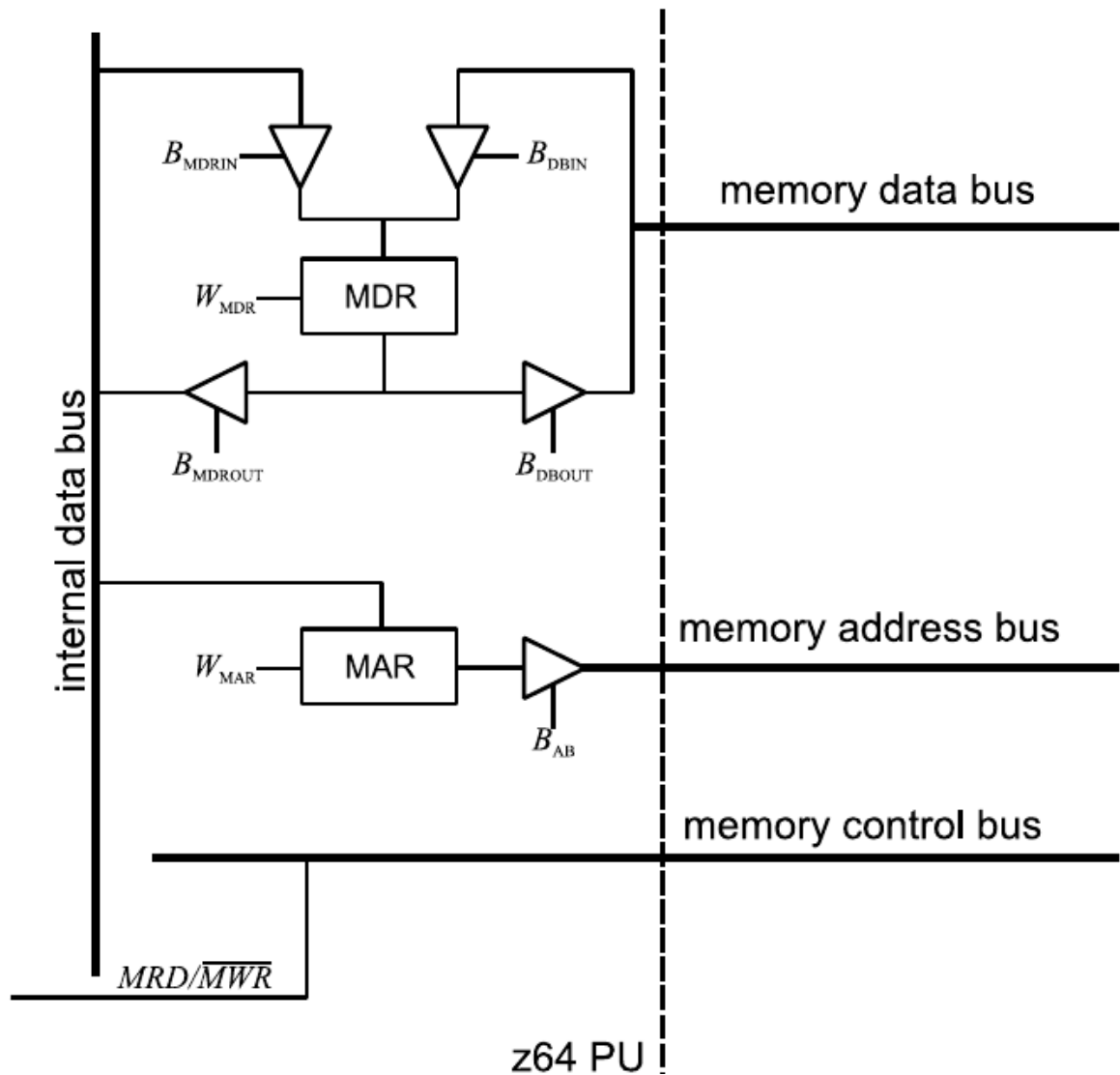
Memoria comportamento esterno

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

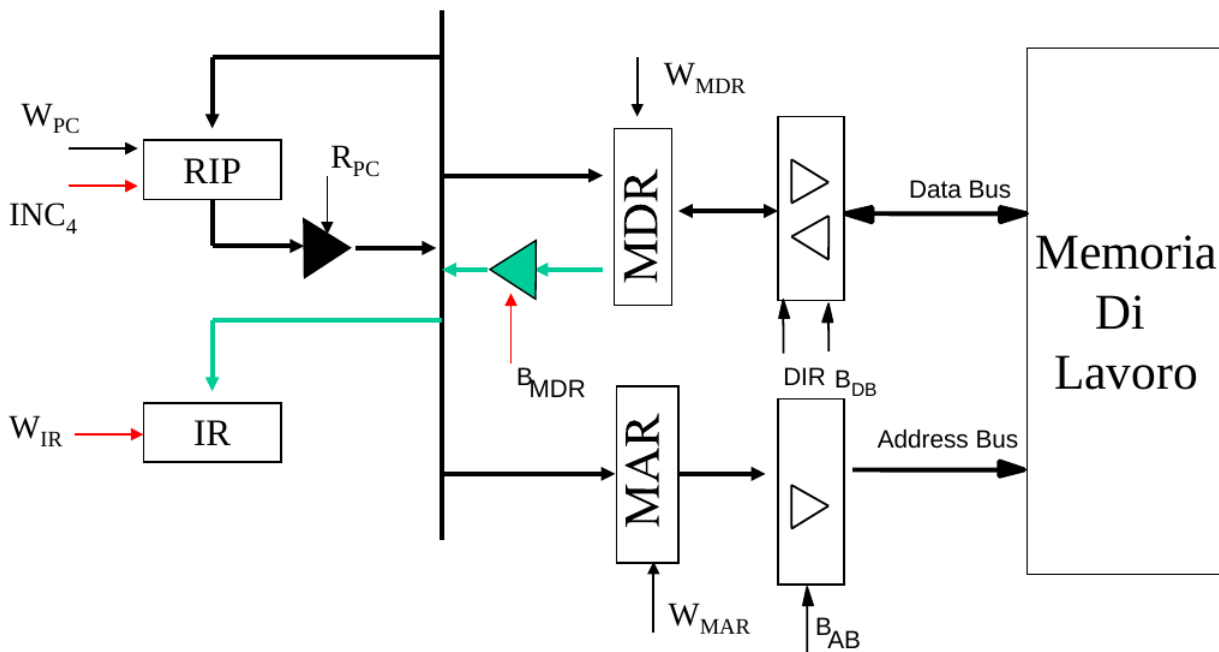


Memoria: interfaccia dello z64



Fetch

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



Fetch: micro-ordini

1. $RIP \rightarrow MAR$; /* trasferimento del contenuto del PC sul MAR */
 1. $R_{RIP} = 1, W_{MAR} = 1$
2. $(MAR) \rightarrow MDR$ /* 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. $MDR \rightarrow IR$ /* trasferimento istruzione da eseguire in IR e predisposizione PC per prelievo prossima istruzione*/
 1. $B_{MDR} = 1, W_{IR} = 1, INC_8 = 1$

Esempio Add

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

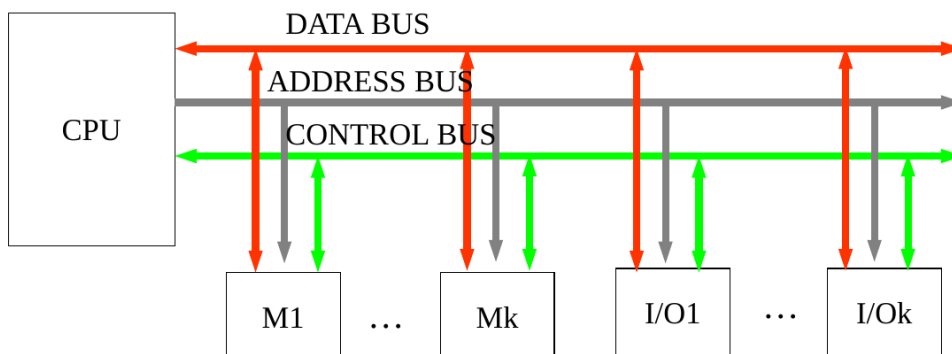
Uno degli operandi (0x20) è memorizzato nei due byte successivi a quelli contenente l'istruzione (indirizzamento immediato)

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

Bus Z64

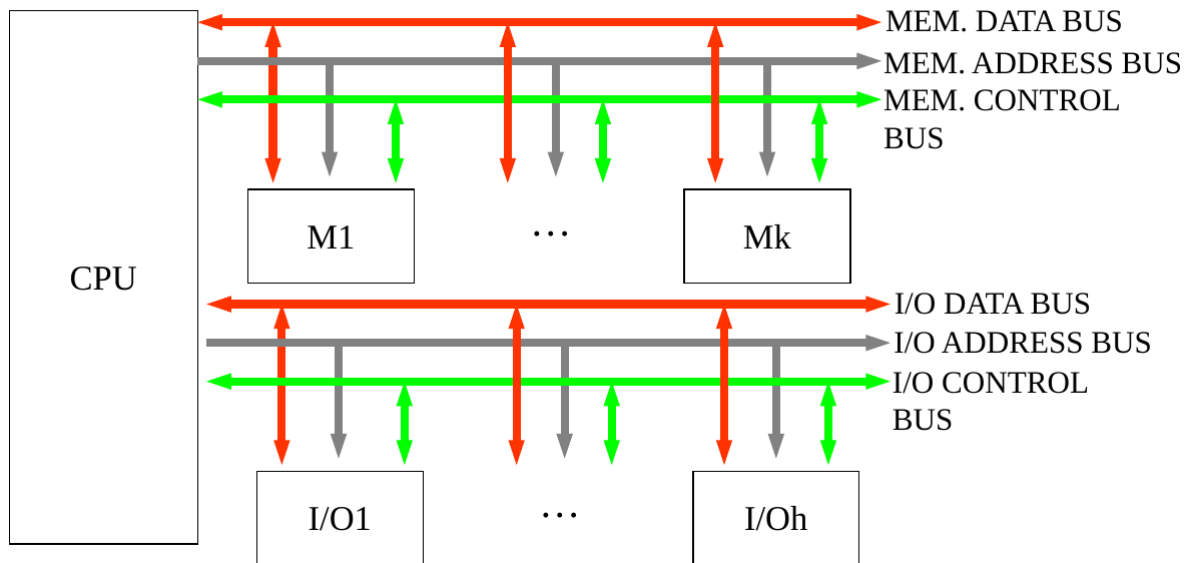
Possibili connessione tra CPU e dispositivi di Ingresso/Uscita

Architettura ad un solo bus



Z64 a due bus

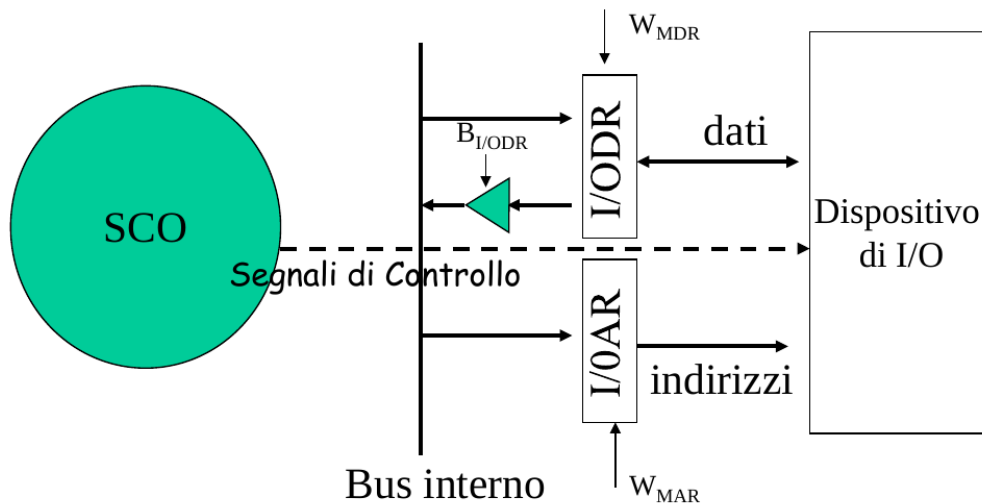
Architettura a due bus: bus di memoria distinto dal bus di I/O



Interfaci dello Z64

Dispositivi di I/O: interfaccia dello z64

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

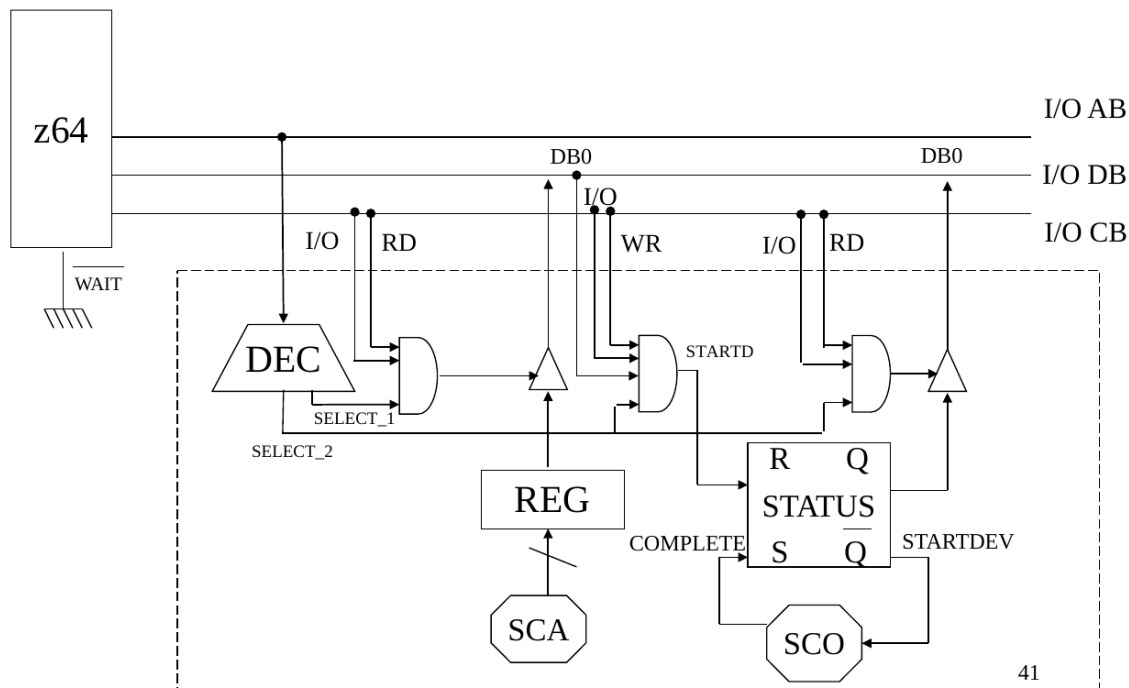


Porte Logiche: Totem Pole vs Open Collector

- E' possibile distinguere due tipologie di porte logiche in funzione dello schema circuitale che le implementa:
 - Totem Pole:
 - In caso di uscita logica “alta”, un transistorore di *pull-up attivo* che forza un livello di tensione alto sul pin d'uscita.
 - In caso di uscita logica “bassa”, un transistorore di *pull-down* che forza un livello di tensione basso sul pin d'uscita.
 - Open Collector:
 - In caso di uscita logica “alta”, l'uscita della porta va in alta impedenza, disconnettendosi dal circuito.
 - In caso di uscita logica “bassa”, la tensione sul pin d'uscita vale 0 (il pin d'uscita è messo a massa)

30

Interfaccia



Operazioni di I/O gestite da canale

La maggior parte delle interazioni tra un dispositivo di Ingresso/Uscita e il processore avviene per trasferire dati (file). Non essendoci grosse necessità elaborative è sufficiente utilizzare dei dispositivi (canali) capaci solo di effettuare il trasferimento di file.

La tecnica utilizzata per far ciò è la Direct Memory Access e il dispositivo che la supporta normalmente viene identificato con DMAC (Direct Memory Access Controller)

DMAC

Per effettuare il trasferimento di un file dalla memoria ad un dispositivo di Ingresso/Uscita o viceversa è necessario definire da processore:

- la direzione del trasferimento (verso o dalla memoria);
- l'indirizzo iniziale della memoria;
- il tipo di formato dei dati (B, W, L), se previsti più formati;
- la lunghezza del file (numero di dati);
- la periferica di Ingresso/Uscita interessata al trasferimento (se ce ne sono più di una).

Utilizzo di un DMAC

Una volta che il DMAC è stato programmato il processore lo deve attivare (p.e. tramite una START)

Da notare che il DMAC per poter trasferire i dati deve poter utilizzare il bus del processore, per questo quando lo usa il processore deve avere le proprie uscite verso il bus in **alta impedenza**.

Una volta che il DMAC ha effettuato il trasferimento dei dati così come richiesto dalla CPU la deve avvertire (p.e. tramite INTERRUPT).

L'architettura di massima del DMAC e il protocollo di interazione processore-DMAC sono schematizzati nei lucidi successivi.

MicroOperazioni

1) **FETCH:**

MAR \leftarrow RIP:
 $B_{RIP} = 1, W_{MAR} = 1$
MDR \leftarrow (MAR), RIP \leftarrow RIP + 8:
 $INC_{RIP} = 1, B_{AB} = 1$
 $B_{AB} = 1, MRD = 1$
 $B_{AB} = 1, MRD = 1, B_{DBIN} = 1, W_{MDR} = 1$
IR \leftarrow MDR:
 $W_{IR} = 1, B_{MDROUT} = 1$

Istruzione **MOV**

movq %rax, %rcx:
 ◦ MAR \leftarrow RIP
 ◦ MDR \leftarrow (MAR); RIP \leftarrow RIP + 8
 ◦ IR \leftarrow MDR
 ◦ TEMP2 \leftarrow RAX
 $S_{MUX} = 1, REG_{MUX} = 0, R_M = 1, W_{T2} = 1$
 ◦ RCX \leftarrow TEMP2
 $S_{MUX} = 2, REG_{MUX} = 1, S_{opcode} = 000000, B_S = 1, W_M = 1$

FULL ADDRESSING

- movq %rax, 0xaaaa(%rax, %rcx, 8):
 - MAR \leftarrow RIP
 - MDR \leftarrow (MAR); RIP \leftarrow RIP + 8
 - IR \leftarrow MDR
 - TEMP2 \leftarrow RCX
 $S_{MUX} = 0, REG_{MUX} = 2, R_M = 1, W_{T2} = 1$
 - TEMP1 \leftarrow Shifter Out[SHL, 000011]
 $S_{opcode} = 000011, B_S = 1, W_{T1} = 1$
 - TEMP2 \leftarrow RAX
 $S_{MUX} = 2, REG_{MUX} = 1, R_M = 1, W_{T2} = 1$
 - MAR \leftarrow ALU OUT[ADD]
 $A_{opcode} = 0000, B_A = 1, W_{MAR} = 1$
 - TEMP1 \leftarrow IR[0:31]
 $B_{SHORT} = 1, W_{T1} = 1$
 - TEMP2 \leftarrow MAR
 $B_{MAROUT} = 1, W_{T2} = 1$
 - MAR \leftarrow ALU OUT[ADD]
 $A_{opcode} = 0000, B_A = 1, W_{MAR} = 1$
 - MDR \leftarrow RAX
 $S_{MUX} = 1, REG_{MUX} = 0, R_M = 1, B_{MDRIN} = 1, W_{MDR} = 1$
 - (MAR) \leftarrow MDR
 $B_{AB} = 1$
 $B_{AB} = 1, MWR = 1$
 $B_{AB} = 1, MWR = 1, B_{DBOUT} = 1$

Istruzioni di movimento dati: immediati ‘piccoli’

- `movl $0xaaaa, %eax:`
 - $MAR \leftarrow RIP$
 - $MDR \leftarrow (MAR); RIP \leftarrow RIP + 8$
 - $IR \leftarrow MDR$
 - $EAX \leftarrow IR[0:31]$
 $S_{MUX} = 2, REG_{MUX} = 1, W_M = 1, B_{SHORT} = 1$

Istruzioni di movimento dati: immediati ‘grandi’

- `movq $0xaaaa, %rax:`
 - $MAR \leftarrow RIP$
 - $MDR \leftarrow (MAR); RIP \leftarrow RIP + 8$
 - $IR \leftarrow MDR$
 - $MAR \leftarrow RIP$
 $B_{rip} = 1, W_{MAR} = 1$
 - $MDR \leftarrow (MAR); RIP \leftarrow RIP + 8$
 $INC_{RIP} = 1, B_{AB} = 1$
 $B_{AB} = 1, MRD = 1$
 $B_{AB} = 1, MRD = 1, B_{DBIN} = 1, W_{MDR} = 1$
 - $RAX \leftarrow MDR$
 $S_{MUX} = 2, REG_{MUX} = 1, W_M = 1, B_{MDROUT} = 1$

Istruzione ADD

- `addw %ax, %cx:`
 - $MAR \leftarrow RIP$
 - $MDR \leftarrow (MAR); RIP \leftarrow RIP + 8$
 - $IR \leftarrow MDR$
 - $TEMP1 \leftarrow AX$
 $S_{MUX} = 1, REG_{MUX} = 0, R_M = 1, W_{T1} = 1$
 - $TEMP2 \leftarrow CX$
 $S_{MUX} = 2, REG_{MUX} = 1, R_M = 1, W_{T2} = 1$
 - $CX \leftarrow ALU\ OUT[ADD]$
 $Aopcode = 0000, B_A = 1, S_{MUX} = 2, REG_{MUX} = 1, W_M = 1, B_{FLAGSA} = 1, W_{FLAGS} = 1$

Le istruzioni SUB sono come le add, semplicemente l'aluopcode non sarà settato, quando chiamato in causa, su add ma su sub

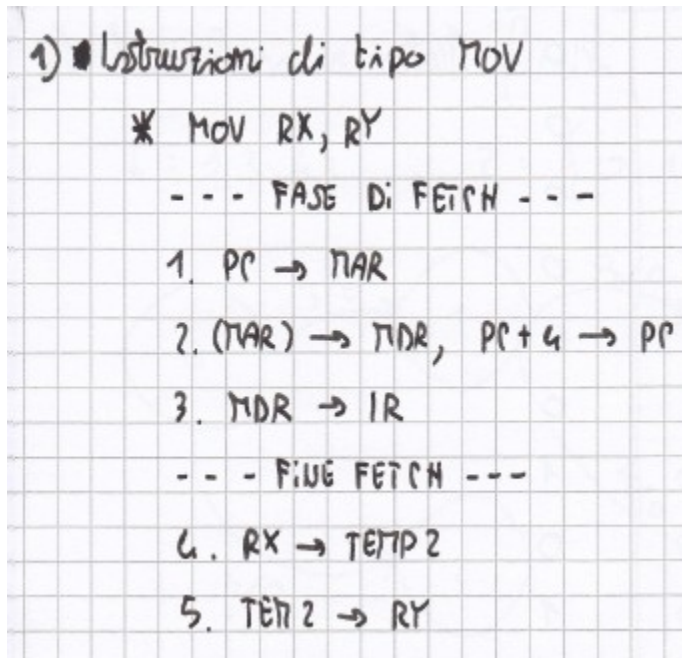
Istruzione AND

- `andw %ax, %cx:`
 - $MAR \leftarrow RIP$
 - $MDR \leftarrow (MAR); RIP \leftarrow RIP + 8$
 - $IR \leftarrow MDR$
 - $TEMP1 \leftarrow AX$
 $S_{MUX} = 1, REG_{MUX} = 0, R_M = 1, W_{T1} = 1$
 - $TEMP2 \leftarrow CX$
 $S_{MUX} = 2, REG_{MUX} = 1, R_M = 1, W_{T2} = 1$
 - $CX \leftarrow ALU\ OUT[AND]$
 $Aopcode = 0110, B_A = 1, S_{MUX} = 2, REG_{MUX} = 1, W_M = 1, B_{FLAGSA} = 1, W_{FLAGS} = 1$

Istruzione di salto condizionale

- jz displacement:
 - $MAR \leftarrow RIP$
 - $MDR \leftarrow (MAR); RIP \leftarrow RIP + 8$
 - $IR \leftarrow MDR$
 - IF $FLAGS[ZF] == 1$ THEN
 - $TEMP1 \leftarrow RIP$
 $B_{RIP} = 1, W_{T1} = 1$
 - $TEMP2 \leftarrow IR[0:31]$
 $B_{SHORT} = 1, W_{T2} = 1$
 - $RIP \leftarrow ALU\ OUT[ADD]$
 $Aopcode = 0000, B_A = 1, W_{RIP} = 1$
 - ENDIF

Esercizi da anni scorsi



II Aggiungiamo ora i TIRRO SEGNALI:

1. $PC \rightarrow MAR$

$BPC = 1$; $WMAR = 1$ ($BPC = \text{BUFFER THREE STATES } PC$)

2. $(MAR) \rightarrow MDR$, $PC + 4 \rightarrow PC$

$BAB = 1$; $INCP4 = 1$ ($BAB = \text{BUFFER THREE STATES } MAR$)

$BAB = 1$; $MEMRD = 1$

$BAB = 1$; $MEMRD = 1$; $BTMDRDBLN = 1$; $WMDR = 1$

3. $MDR \rightarrow IR$

$BTMDRISOUT = 1$; $WIR = 1$

4. $RX \rightarrow TEMP2$

$SELMUX = 0$; $BR = 1$; $WTZ = 1$

5. $TEMP2 \rightarrow RT$

$SELMUX = 1$; $SHIFTER_OPCODE[000000] = 1$; $WR = 1$; $BS = 1$

Altri esempi

* MOV (RX), RY

- - - FETCH - - -

1. PC \rightarrow MAR

BPC = 1 ; WMAR = 1

2. (MAR) \rightarrow MDR ; PC + 4 \rightarrow PC

BAB = 1 ; INCP4 = 1

BAB = 1 ; MRD = 1

BAB = 1 ; MRD = 1 ; BMDRDBLN = 1 ; WMDR = 1

3. MDR \rightarrow IR

BMDRDBOUT = 1 ; WLR = 1

- - - FINE FETCH - - -

4. RX \rightarrow MAR

SELNEX = 0 ; BR = 1 ; WMAR = 1

5. (MAR) \rightarrow MDR

BAB = 1

BAB = 1 ; MRD = 1

BAB = 1 ; MRD = 1 ; BMDRDBLN = 1 ; WMDR = 1

6. MDR \rightarrow RY

SELNEX = 1 ; BMDRDBOUT = 1 ; WR = 1

Altri esempi

* MOV RX, (RY)

4. RX → MAR

SELNEX = 1; BR = 1; WMAR = 1

5. ● RY → MDR

SELNEX = 0; BR = 1; BMDRBLN = 1; WMRD = 1

6. MDR → (MAR)

BAB = 1; BMDRDBOUT = 1

" ; " ; MEMWR = 1

* MOV (RX), (RY)

4. RX → MAR

SELNEX = 0; BR = 1; WMAR = 1

5. MAR → MDR

BAB = 1

BAB = 1; MMRD = 1

BAB = 1; MMRD = 1; BMDRDBLN = 1; WMDR = 1

6. RY → MAR

SELNEX = 1; BR = 1; WMAR = 1

7. MDR → (MAR)

BAB = 1; BMDRDBOUT = 1

" ; " ; MEMWR = 1

* MOV ADDRESS, RY

4. PC \rightarrow MAR

BPC = 1; WMAR = 1

5. (MAR) \rightarrow MDR

BAB = 1

BAB = 1; MRD = 1

BAB = 1; MRD = 1; BMDRDBLN = 1; WMDR = 1

6. PC + 4 \rightarrow PC

INCP4 = 1

7. BMDRDBOUT = 1; WMAR = 1
MDR \rightarrow MAR

8. REMPLA PASSO 5

9. MDR \rightarrow RY

SELUX = 1; BMDRDBOUT = 1; WR = 1

* MOV Rx, ADDRESS

// Identica alla precedente fino al passo 7.

8. ~~PC~~ Rx \rightarrow MDR

SELUX = 0; BR = 1; BMDRDBLN = 1

9. MDR \rightarrow (MAR)

BAB = 1; BMDRDBOUT = 1

11 11 ; MEMWR = 1

Istruzioni esempi add

* ADD RX, RY

4. RX → TEMP1

BR = 1; SELTUX = 0; WT1 = 1

5. RY → TEMP2

BR = 1; SELTUX = 1; WT2 = 2

6. ALU_OUT(ADD) → RY

WR = 1; SELTUX = 1; BA = 1; OPCODEALU = ADD; BTEMP2 = 1

* ADD (RX), RY

4. RX → PAR

BR = 1; SELTUX = 0; WPAR = 1

5. (PAR) → TDR

BAB = 1;

BAB = 1; MEMRD = 1

BAB = 1; MEMRD = 1; BMDRDBLU = 1; WMDR = 1

6. TDR → TEMP1

BMDRUBOUT = 1; WT1 = 1

7. RY → TEMP2

BR = 1; SELTUX = 1; WT2 = 1

8. ALU → RY

WR = 1; SELTUX = 1; BTEMP = 2; BA = 1; ALU OPCODE = ADD

Esempi MOV

* $MOV(RX)+, RY$

I primi sei passi sono identici a $(RX), RY$; il settimo è:

7. $S \rightarrow ALU$; $ALU_OUT(ADD) \rightarrow RX$

$SELNUX = 1$; $BA = 1$; $WR = 1$; $SELNUX = 0$; $ALU_OPCODE = ADD$

* $MOV-(RX), RY$

4. $RX \rightarrow TEMP1$

$SELNUX = 0$; $BR = 1$; $WIT1 = 1$

5. $S \rightarrow ALU$; $ALU_OUT(SUB) \rightarrow RX$; $ALU_OUT(SUB) \rightarrow MAR$

$WMAR = 1$; $BA = 1$; $WR = 1$; $SELNUX = 0$; $OPCODE_ALU = SUB$

6. $MAR \rightarrow MDR$

7. $MDR \rightarrow RY$

* $MOV\ RX, (RY)+$

4. $RY \rightarrow MAR$; $RY \rightarrow TEMP1$

$SELNUX = 1$; $BR = 1$; $WMAR = 1$; $WIT1 = 1$

5. $RX \rightarrow MDR$

$SELNUX = 0$; $BR = 1$; $BMDRBLW = 1$

6. $MDR \rightarrow MAR$

7. $S \rightarrow ALU$; $ALU_OUT[ADD] \rightarrow R1$

* $MOV\ RX, -(RY)$

4. $RY \rightarrow TEMP1$

5. $S \rightarrow ALU$; $ALU_OUT(SUB) \rightarrow RT$; $ALU_OUT(SUB) \rightarrow RY$

6. $RX \rightarrow MDR$

7. $MDR \rightarrow (MAR)$

Esercizi sui sistemi lineari

* Algoritmo per convertire in forma ESPONENTE/MANTISSA.

1. Si converte in binario la parte intera
2. Si converte in binario la parte decimale
3. Si scrive il numero così ottenuto in forma completa
4. Si shifta a sinistra la virgola fino a quando si ottiene un numero nella forma $1,xyz\dots$; ciò che resta dopo la virgola è la MANTISSA
5. L'esponente è pari a $127 + \text{numero di shift}$; lo si converte quindi in binario
6. Il numero finale è a 32 bit, nel seguente ordine:
 - * 1° bit: segno
 - * dal 2° al 9° bit: esponente
 - * dal ~~decimo~~ ~~decimo~~ 10° al 32° bit: mantissa completata da eventuali 0

* Algoritmo ~~da~~ da ~~esponente~~ virgola mobile a decimale

* Semplicemente, prendere i bit 4 a 4 e convertirli in esadecimale

* Algoritmo da virgola mobile a decimale

1. Il primo bit ci dà il segno
2. Calcolare E come $E - 127$
3. Normalizzare la mantissa come $1 + \text{conversione mantissa in decimale}$
4. Il numero è quindi $2^E \cdot \text{mantissa convertita}$

Distanza di Hamming

La **distanza di Hamming** $d(x,y)$ fra due parole è il **numero di posizioni** (bit) per cui esse differiscono. Tale distanza può essere utile per definire quando un codice è ambiguo, ridondante o irridondante.

Ex. $d(10010, 01001) = 4$

La distanza minima di un codice è data dalla minima fra le distanze minime

$$d_{\min} = \min(d(x,y))$$

Se $h = 1$ (e $n = m$) si ha un codice irridondante

se $h > 1$ (e $n > m$) si ha un codice ridondante

Se $h = 0$ si ha un codice ambiguo.



Esempi di calcolo distanza di Hamming

Parole di C	Prima codifica	Seconda codifica	Terza codifica	Quarta codifica	Quinta codifica
alfa	000	0000	00	0000	110000
beta	001	0001	01	0011	100011
gamma	010	0010	11	0101	001101
delta	011	0011	10	0110	010110
mu	100	0100	00	1001	011011

$h = 1$

Irr.

$h = 1$

Rid.

$h = 0$

Amb.

$h = 2$

Rid.

Rivela

errori

$h = 3$

Rid.

Rivela

e corregge

errori

Esistono diversi tipi di codice. I codici carattere vengono usati per rappresentare in binario i simboli non numerici. Il codice più diffuso è il codice **ASCII**.

La rilevazione di errori di trasmissioni viene solitamente eseguita introducendo ridondanza nelle informazioni trasmesse. Su una trasmissione di un codice (n, k) ci saranno n bit trasmessi per k bit di informazioni, con $n > k$.

Il **peso di un errore** rappresenta il numero di bit che sono stati modificati durante la trasmissione.

Un codice a distanza minima d è capace di rilevare errori di peso $\leq d - 1$

Codice di parità

Il codice di parità è un codice in cui si inserisce un bit che vale 0 quando il numero di 1 è pari e che vale 1 quando è dispari.

Ex. 0**1010001** **1**, poiché il numero di 1 dispari

Ex. 0**1111000** **0**, poiché il numero di 1 è pari

Il codice di parità può essere ottenuto dalle seguenti espressioni

$$b_1 + b_2 + b_3 + \dots + b_n + p = 0 \quad \text{parità oppure}$$

$$b_1 + b_2 + b_3 + \dots + b_n + p = 1 \quad \text{disparità}$$

Dove

- **n** è il numero di bit usati per rappresentare in binario gli oggetti (informazione),
- **+** e' l'operatore di **somma modulo 2**
- **p** il bit di "parita/disparità" da aggiungere a quelli di informazione per costruire parole del codice

Bit di informazione	Parità	Disparità
000	000 0	000 1
001	001 1	001 0
010	010 1	010 0
011	011 0	011 1
100	100 1	100 0
101	101 0	101 1
110	110 0	110 1
111	111 1	111 0

Il codice ottenuto è un codice di **distanza minima pari a 2**, cioè in grado di rilevare errori di pesi 1 (single error). Introducendo il bit di parità (o di disparità) è infatti facile vedere come la distanza minima, fra tutti gli elementi del codice, aumenti di 1 e diventi proprio pari a 2.

Codici Hamming

Quello della generazione di codici Hamming è un modo per costruire codici a **distanza minima 3**.

Per ogni i è possibile costruire un codice a $2^i - 1$ bit con i bit di parità e $2^i - 1 - i$ bit di informazione. I bit in posizione corrispondente ad una potenza di 2 sono bit di parità, i rimanenti sono bit di informazione.

Ogni bit di parità controlla la correttezza dei bit di informazione la cui posizione, espressa in binario, ha un 1 nella potenza di 2 corrispondente al bit di parità.

Per capire di cosa si tratta immaginiamo di avere una informazione a 4 bit. Si avrà quindi un codice con 4 bit di informazione. Poiché i bit d'informazione erano pari a $2^i - 1 - i = 4$, si potrà dire che i , ossia il numero di bit di parità, sarà pari a 3.

Poiché i bit di parità hanno come posto la posizione corrispondente ad una potenza di 2, si avrà un bit di parità nelle posizioni 1, 2, 4.

$p_1 \ p_2 \ l_3 \ p_4 \ l_5 \ l_6 \ l_7$

p_1 controlla la parità di $l_3 \ l_5 \ l_7$

p_2 controlla la parità di $l_3 \ l_6 \ l_7$

p_4 controlla la parità di $l_5 \ l_6 \ l_7$

Si procederà quindi con il calcolo dei bit di parità scegliendo quello che verifica le seguenti equazioni

$$p_1 \oplus l_3 \oplus l_5 \oplus l_7 = 0$$

$$p_2 \oplus l_3 \oplus l_6 \oplus l_7 = 0$$

$$p_4 \oplus l_5 \oplus l_6 \oplus l_7 = 0$$

Esempio

Se si vuole trasmettere l'informazione 1100, si avrà $p_1 \ p_2 \ 1 \ p_4 \ 1 \ 0 \ 0$

e, dai calcoli, $p_1 = 0$, $p_2 = 1$, $p_4 = 1$. Quindi l'informazione da inviare sarà **0111100**.

Qualora il codice ricevuto sia **1111100**, basterà verificare se le condizioni sono verificate per identificare l'eventuale presenza di errori. Infatti, in questo caso:

$$p_1 \oplus l_3 \oplus l_5 \oplus l_7 = \mathbf{1}$$

$$p_2 \oplus l_3 \oplus l_6 \oplus l_7 = \mathbf{0}$$

$$p_4 \oplus l_5 \oplus l_6 \oplus l_7 = \mathbf{0}$$

Che letto dall'ultimo al primo ottengo **001**, ossia avrò un errore in posizione **1**.

Nel caso in cui il codice ricevuto sia 01111**10**

$$p_1 \oplus l_3 \oplus l_5 \oplus l_7 = \mathbf{0}$$

$$p_2 \oplus l_3 \oplus l_6 \oplus l_7 = \mathbf{1}$$

$$p_4 \oplus l_5 \oplus l_6 \oplus l_7 = \mathbf{1}$$

Avrò un errore in posizione **110**, ossia in posizione **6**.