Interfacciamento

Processore – dispositivi di I/O

Gestione delle operazioni di I/O

Caratteristiche dei dispositivi di Ingresso/Uscita

Tipo	Codice	Velocità di scambio
Dischi Magnetici	Byte	Fino a 300 Mcar/sec
Nastri Magnetici	Byte	Fino a 30 Mcar/sec
Stampante Seriale	Byte	200 – 1200 car/sec
Stampante Parallela	Byte	1K – 100K car/sec
Terminali CRT	Byte	300 – 19,2K car/sec
Convertitori analogico-digitali	Parola 8-16 bit	10-10M parole/sec
USB 1.0	Byte	1,5Mcar/sec
USB 2.0	Byte	60Mcar/sec

Evoluzione delle prestazioni del sottosistema di I/O

- Incremento Prestazioni CPU: 60% ogni anno
- Le prestazioni dei sistemi di I/O system sono tipicamente (dischi, stampanti etc..) limitate da ritardi *meccanici*
  Incremento prestazioni < 10% ogni anno
- Legge di Amdahl:

$$S = \frac{1}{(1 - f) + \frac{f}{K}}$$

- S è lo speed-up effettivo
- f è la frazione di lavoro non dipendente dall'I/O
- K è lo speed-up della modalità "veloce"

- Lo speed-up di un sistema è limitato dal componente più lento:
  p.e. 10% x I/O & 90% x CPU (f=0,9)
  se K=10 => S prossimo a 5
  se K=1000 => S prossimo a 10
  Il sottosistema di I/O rappresenta un collo di bottiglia

Tipi di interazione tra CPU e dispositivi esterni

- **Previste dai programmi eseguiti nella CPU**
  (I/O programmato)
- **Su richiesta esterna**
- **Gestite da processori dedicati (canali)**

Esempi quotidiani di interazione

I/O programmato:
- controllo della cottura della pasta
- ricevimento studenti

su richiesta esterna:
- ricezione di una telefonata
- ricezione di una lettera

gestite da processori dedicati
- gestione delle comunicazioni tramite un servizio di segreteria (persona dedicata)

Tipi di interazione tra CPU e dispositivi esterni che studieremo

- Previste dai programmi che vengono eseguiti nella CPU (I/O programmato):
  - modalità busy waiting, implementata:
    i. a firmware
    ii. a software
  - modalità polling

- Su richiesta esterna:
  - interruzione
- Gestite da processori dedicati (canali)
  - Direct Memory Access Controller (DMAC)

ISTRUZIONE DI OUTPUT

outX %?a?, %dx

• sposta nel registro di interfaccia il cui indirizzo è memorizzato nel registro %dx quanto è memorizzato nel registro accumulatore (si preleveranno 8, 16, 32 o 64 bit, a secondo del valore di X, specificato nell'istruzione stessa, e quindi i dati potranno essere prelevati da %al, %ax, %eax oppure da %rax).

• da notare che per identificare un registro esterno si utilizzano 16 bit, questo permette uno spazio di indirizzamento di 2^{16} (64K locazioni differenti) che è comunque un valore molto elevato considerando il numero delle periferiche presenti in un sistema di elaborazione.

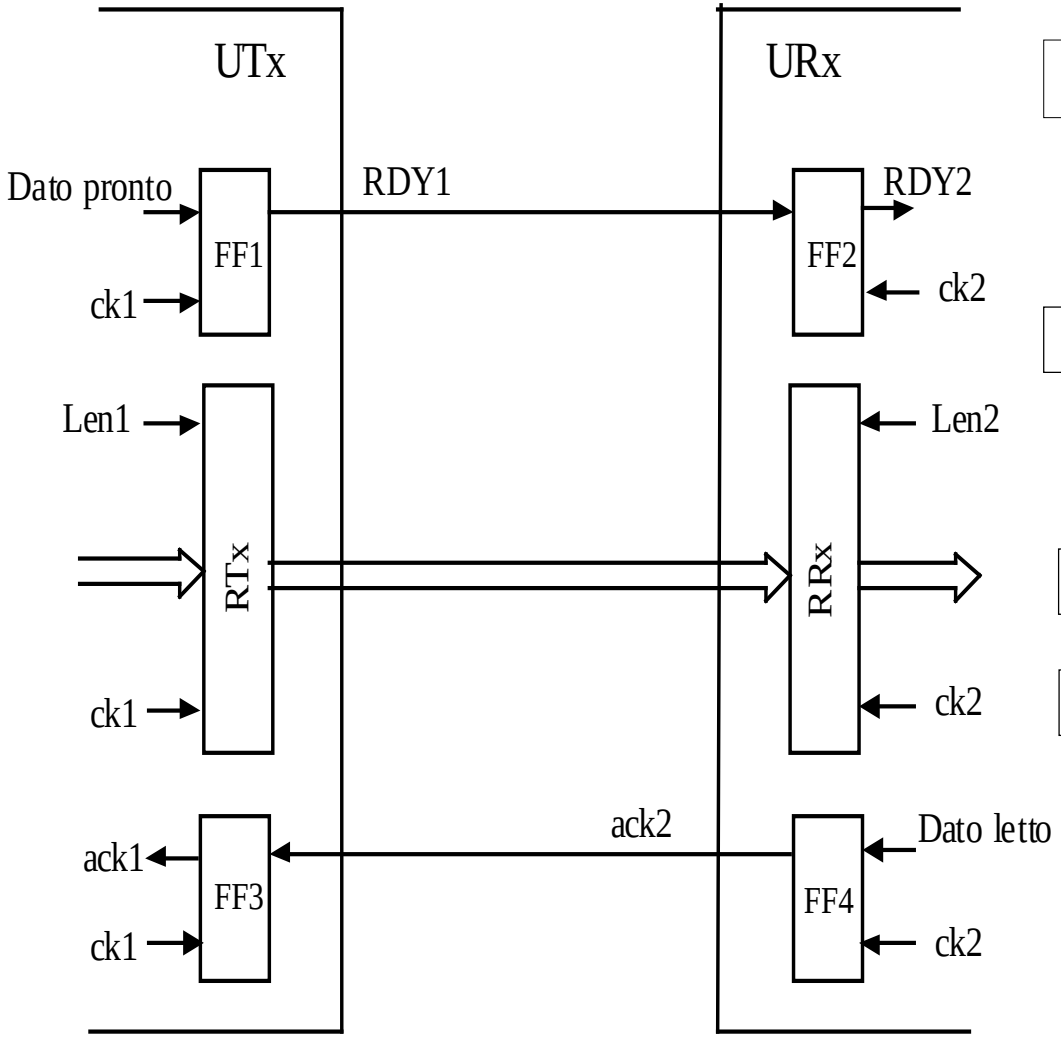
ISTRUZIONE DI INPUT

inX %dx, %?a?

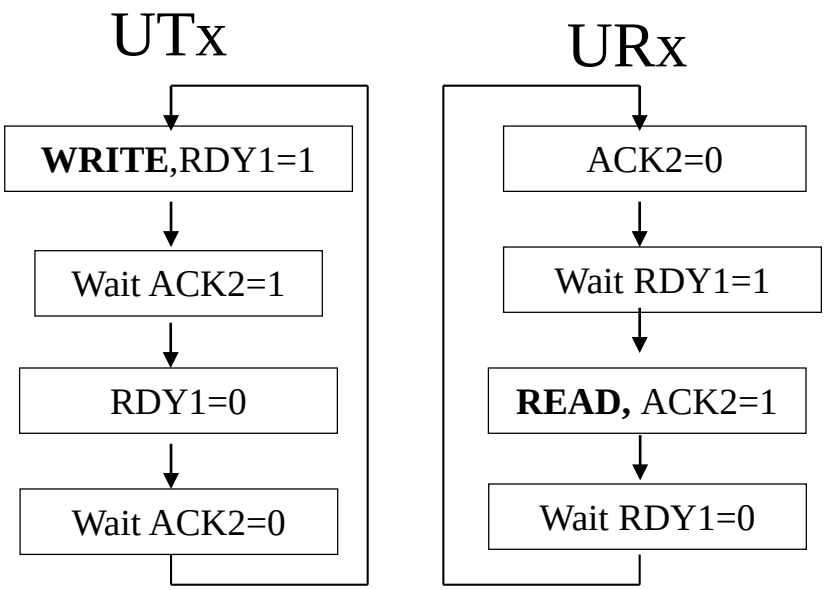
- sposta nell'accumulatore quanto memorizzato nel registro interfaccia della periferica il cui indirizzo è memorizzato nel registro %dx (si preleveranno 8, 16, 32 o 64 bit, a secondo del valore di X, specificato nell'istruzione stessa, e quindi i dati potranno essere memorizzati in %al, %ax, %eax oppure in %rax).

- da notare che per identificare un registro esterno si utilizzano 16 bit, questo permette uno spazio di indirizzamento di 2^{16} (64K locazioni differenti) che è comunque un valore molto elevato considerando il numero delle periferiche presenti in un sistema di elaborazione.

“Richiamo” INTERFACCIA Sistemi Digitali Complessi (per supportare protocollo di handshaking)

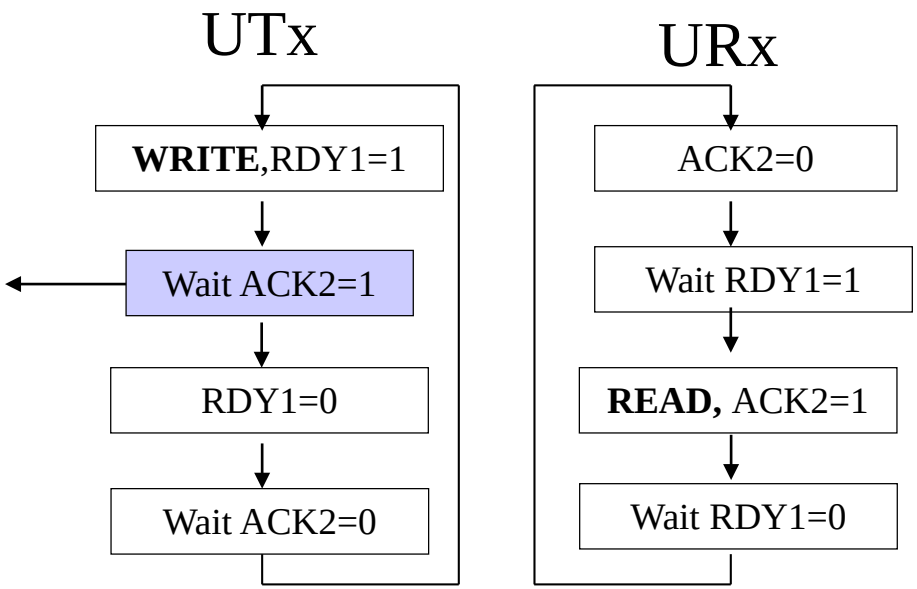


Initial values: RDY1=0, ACK2=0

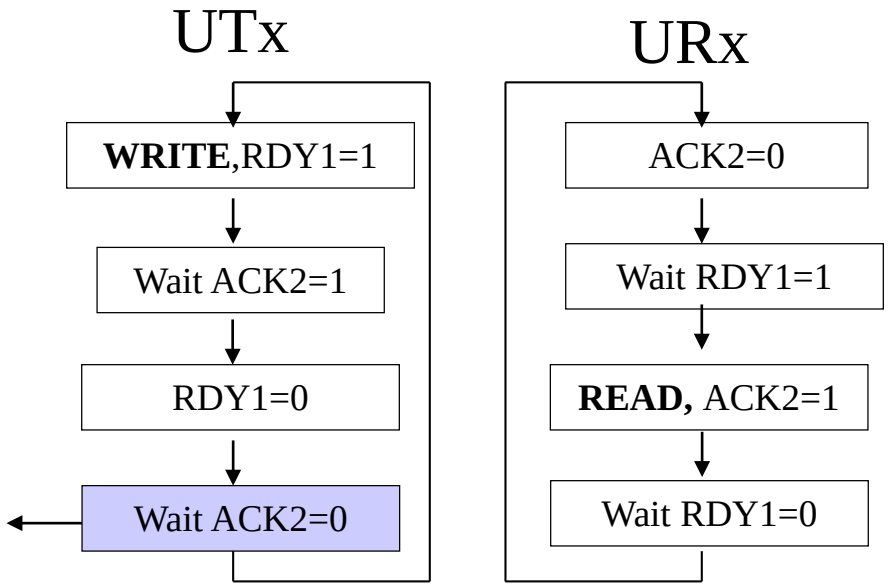


Cosa succederebbe se si rimuovessero le “wait”?

Sovrascrivo prima che la lettura sia avvenuta…

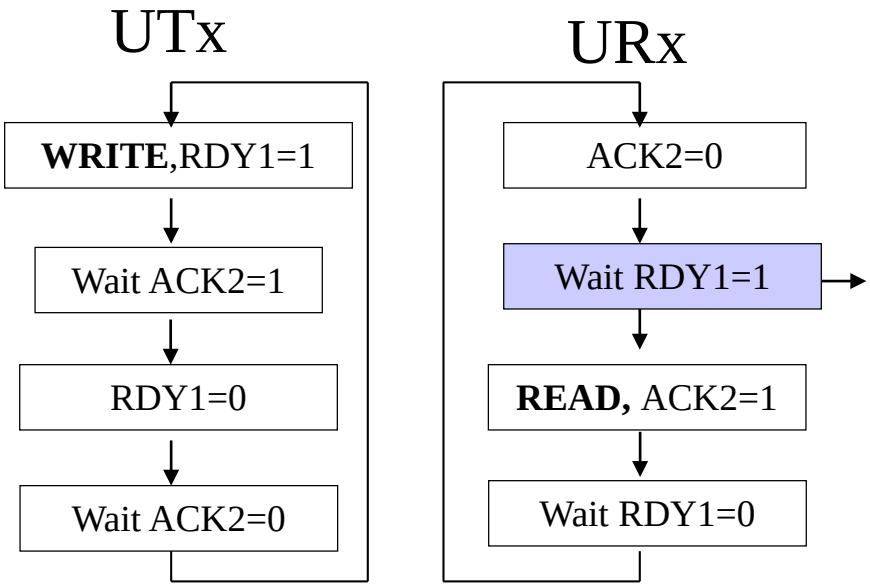


Cosa succederebbe se si rimuovessero le wait?



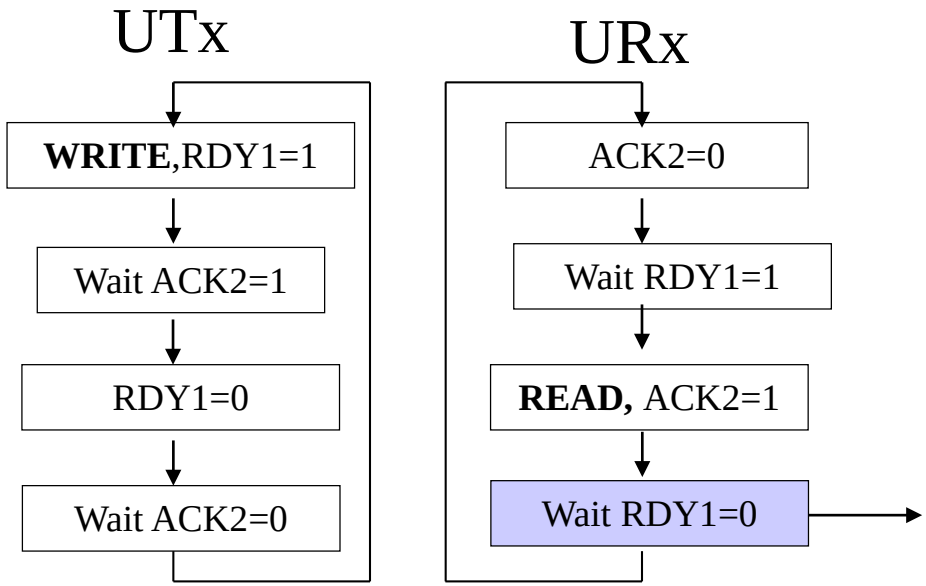
Al ciclo successivo dopo aver scritto potrei trovare ACK=1 dal ciclo precedente, senza che la lettura del nuovo valore sia avvenuta: SOVRASCRIVO!

Che succederebbe se si rimuovessero le wait?



Potrei leggere prima che la scrittura sia avvenuta!

Che succederebbe se si rimuovessero le wait?

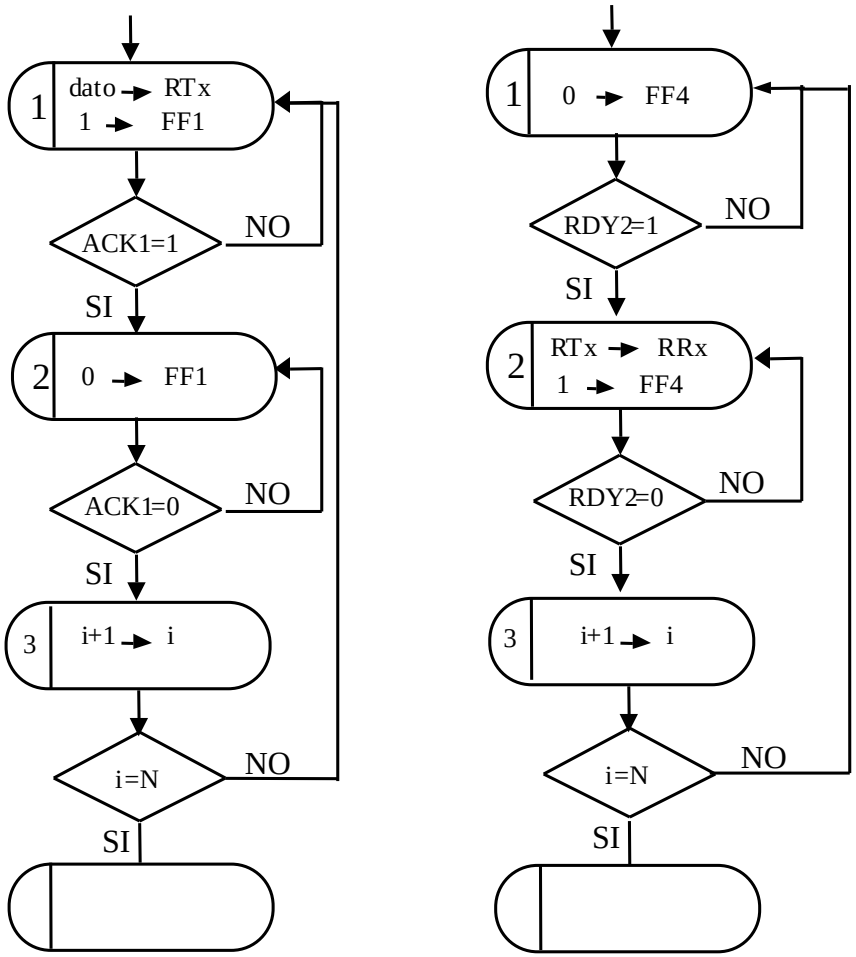


Potrei leggere 2 volte lo stesso valore!

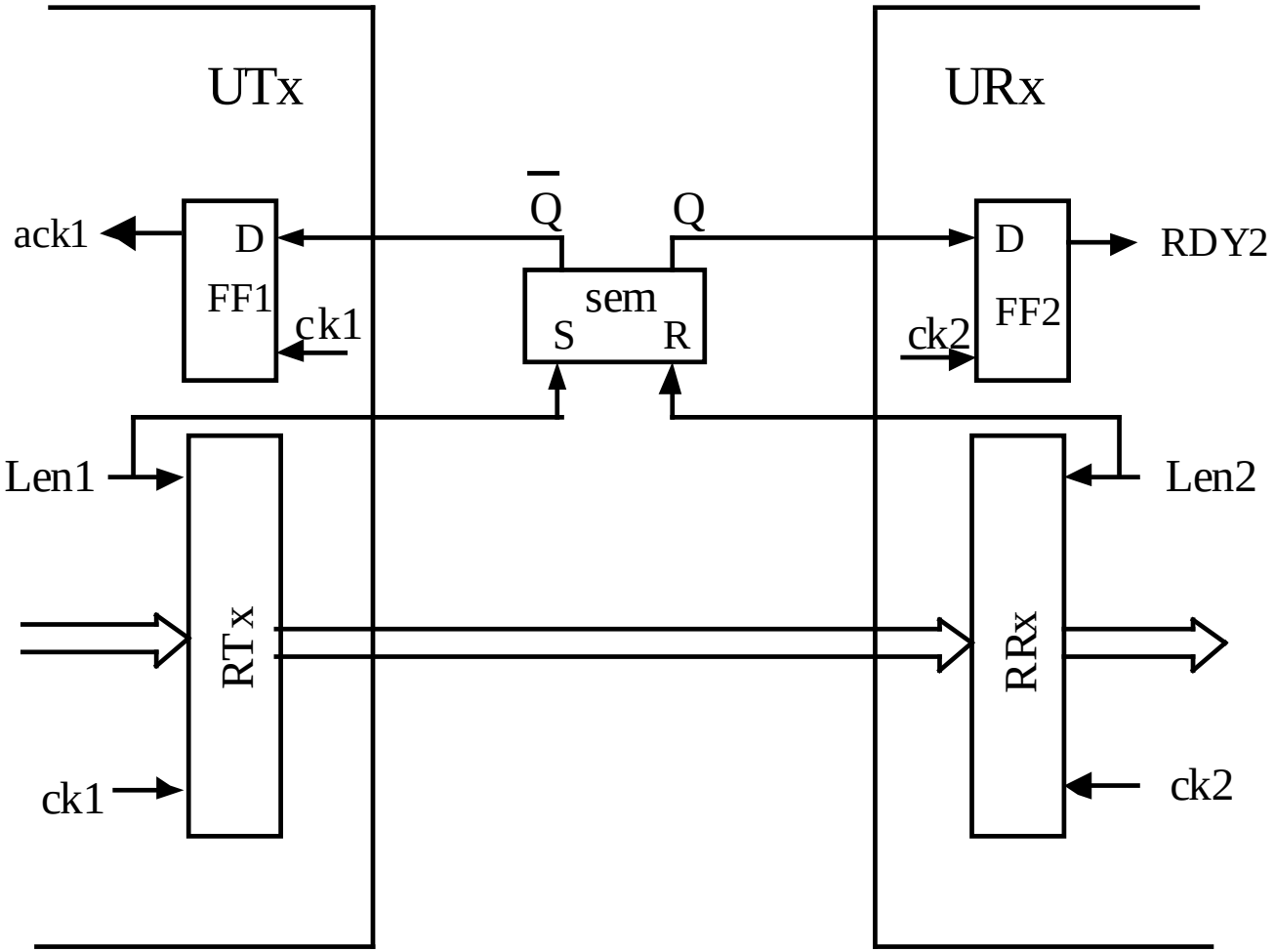
Protocollo di HandShaking per il trasferimento di N dati

Utx	**URx**
1: dato→RTx, 1→FF1;	1: 0→FF4;
2: if ack1=0, then vai a 2;	2: if RDY2=0, then vai a 2;
3: 0→FF1;	3: RTx→RRx, 1→FF4;
4: if ack1=1, then vai a 4;	4: if RDY2=1, vai a 4 ;
5: i+1→i;	5: i+1→i;
6: if i≠N, vai a 1	6: if i≠N, vai a 1

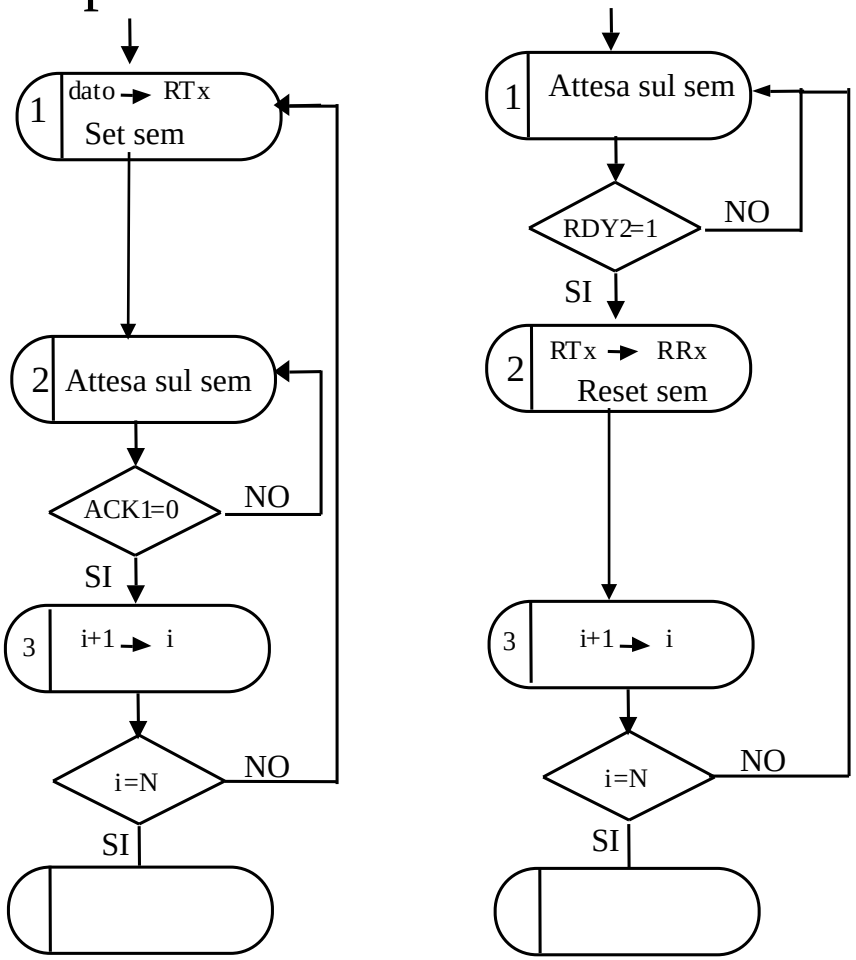
Sequenze di microistruzioni eseguite da UTx e URx durante il protocollo di comunicazione



“Altra” INTERFACCIA di Sistemi Digitali Complessi
(per supportare protocollo di handshaking)



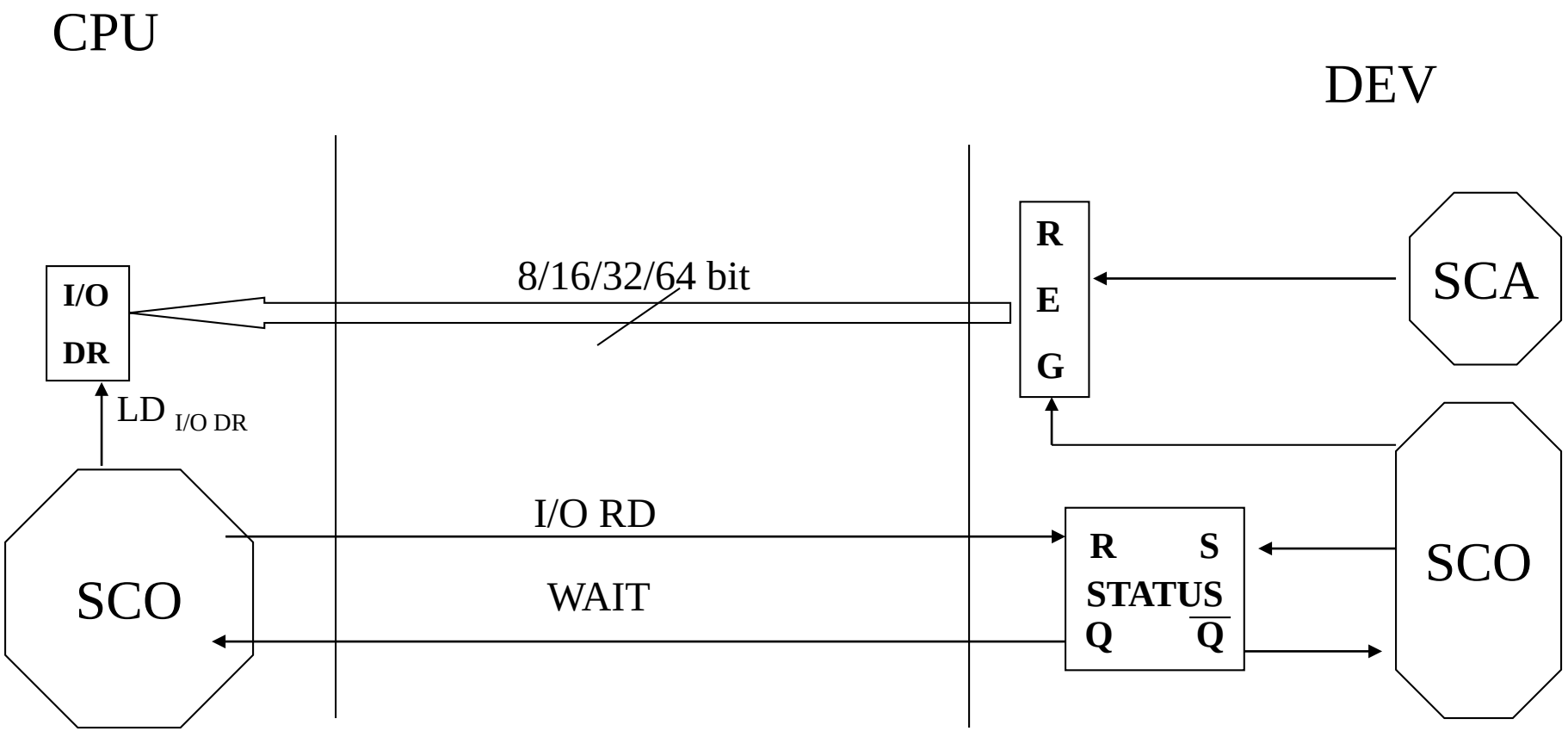
Sequenze di microistruzioni eseguite da UTx e URx durante il protocollo di comunicazione



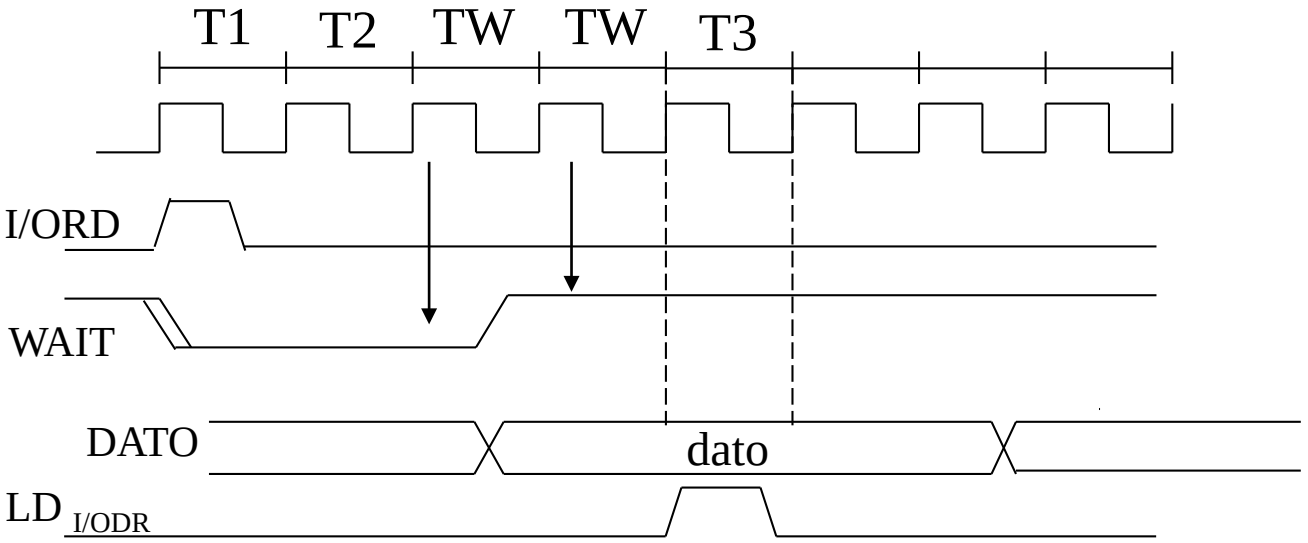
I/O programmato INTERFACCIA DISPOSITIVI di I/O

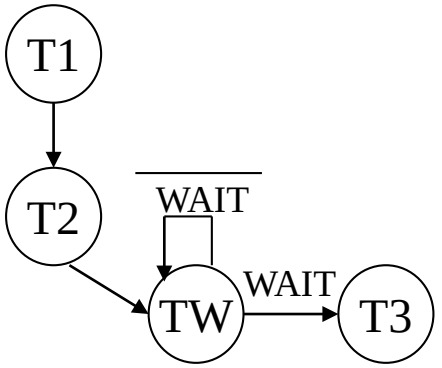
(per supportare protocollo di handshaking, implementato a firmware)

Schema di Interfaccia per l'input tra z64 e un <u>solo</u> Sistema Digitale Complesso



Temporizzazione dei segnali

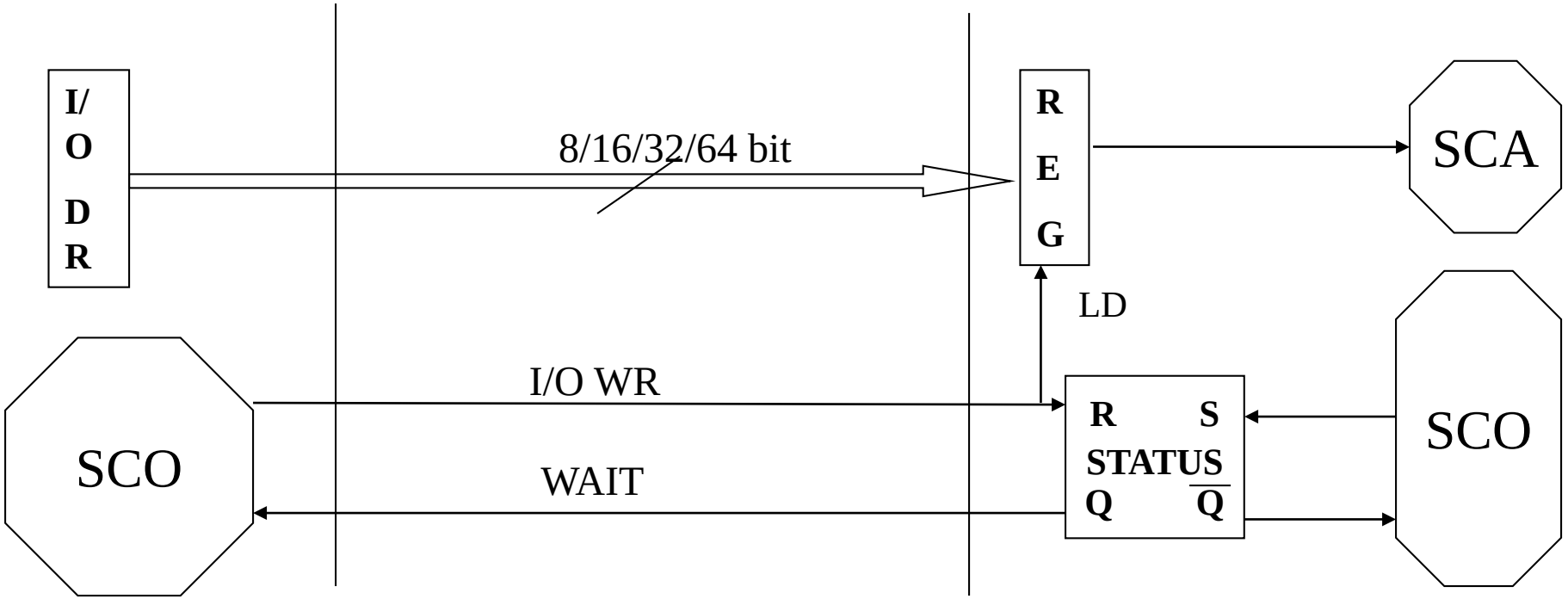




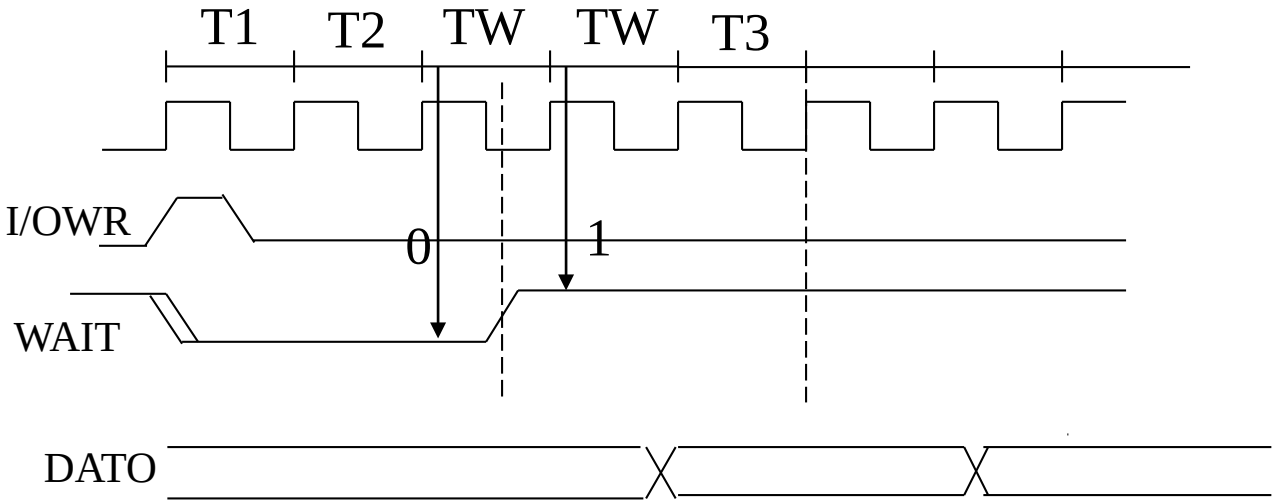
I/O programmato INTERFACCIA DISPOSITIVI DI I/O

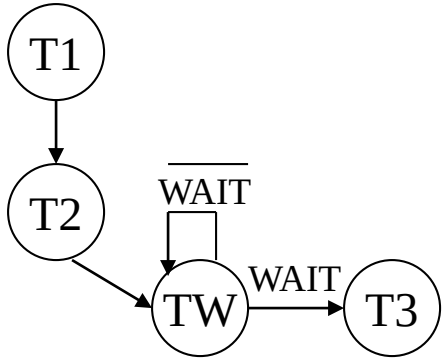
(per supportare protocollo di handshaking, implementato a firmware)

Schema di Interfaccia per l'output tra z64 e un <u>solo</u> Sistema Digitale Complesso



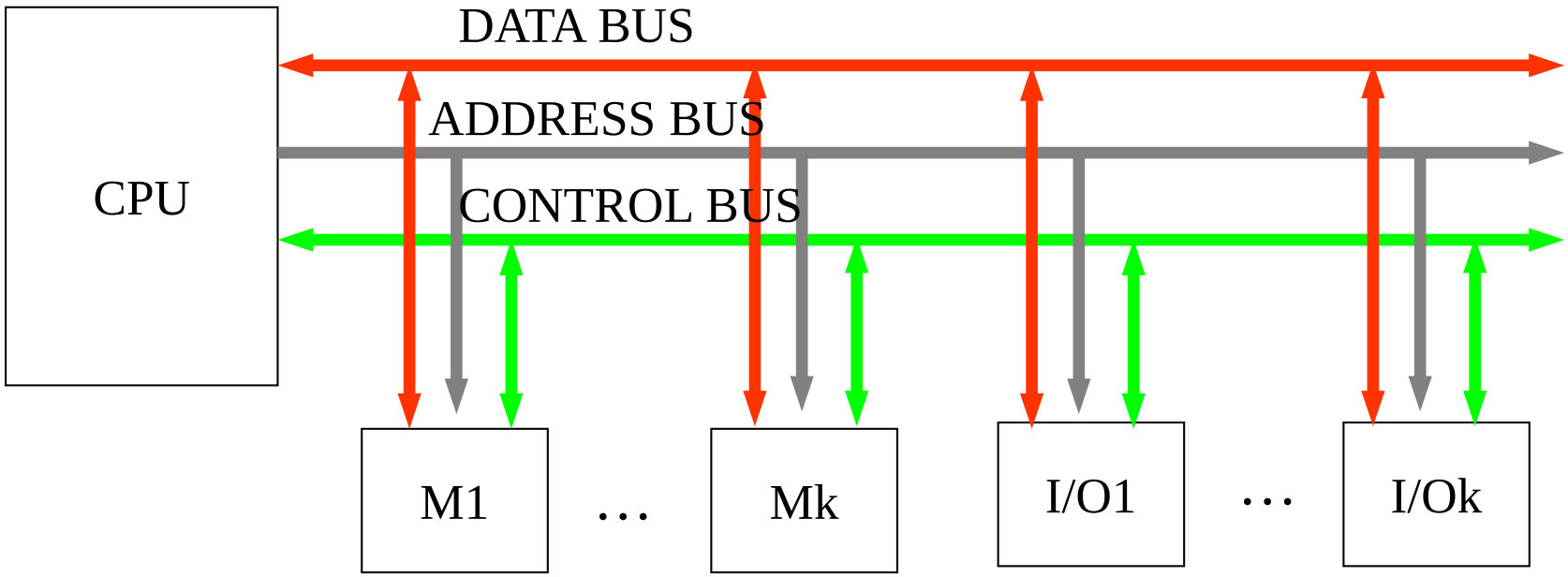
Temporizzazione dei segnali





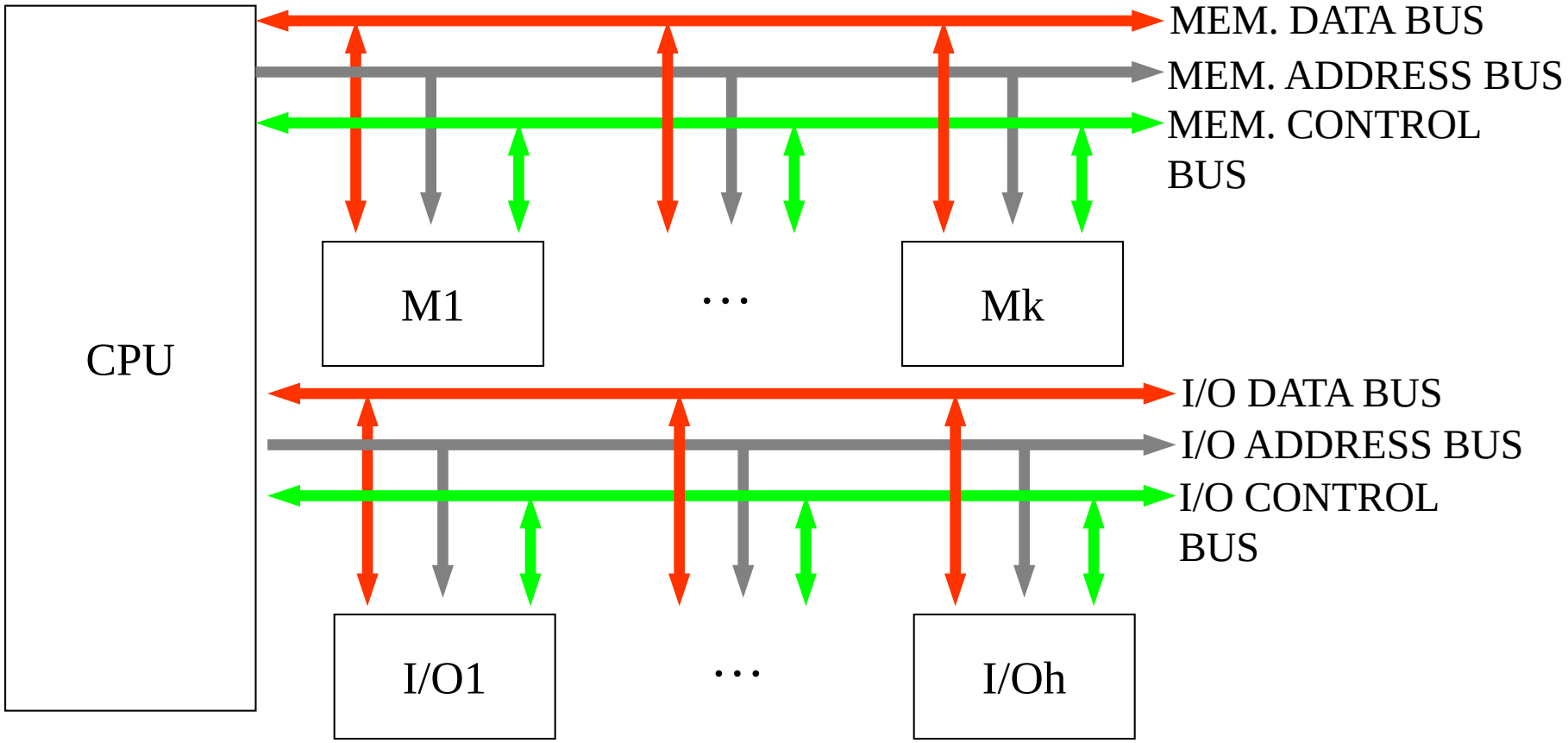
Possibili connessione tra CPU e dispositivi di Ingresso/Uscita

Architettura ad un solo bus



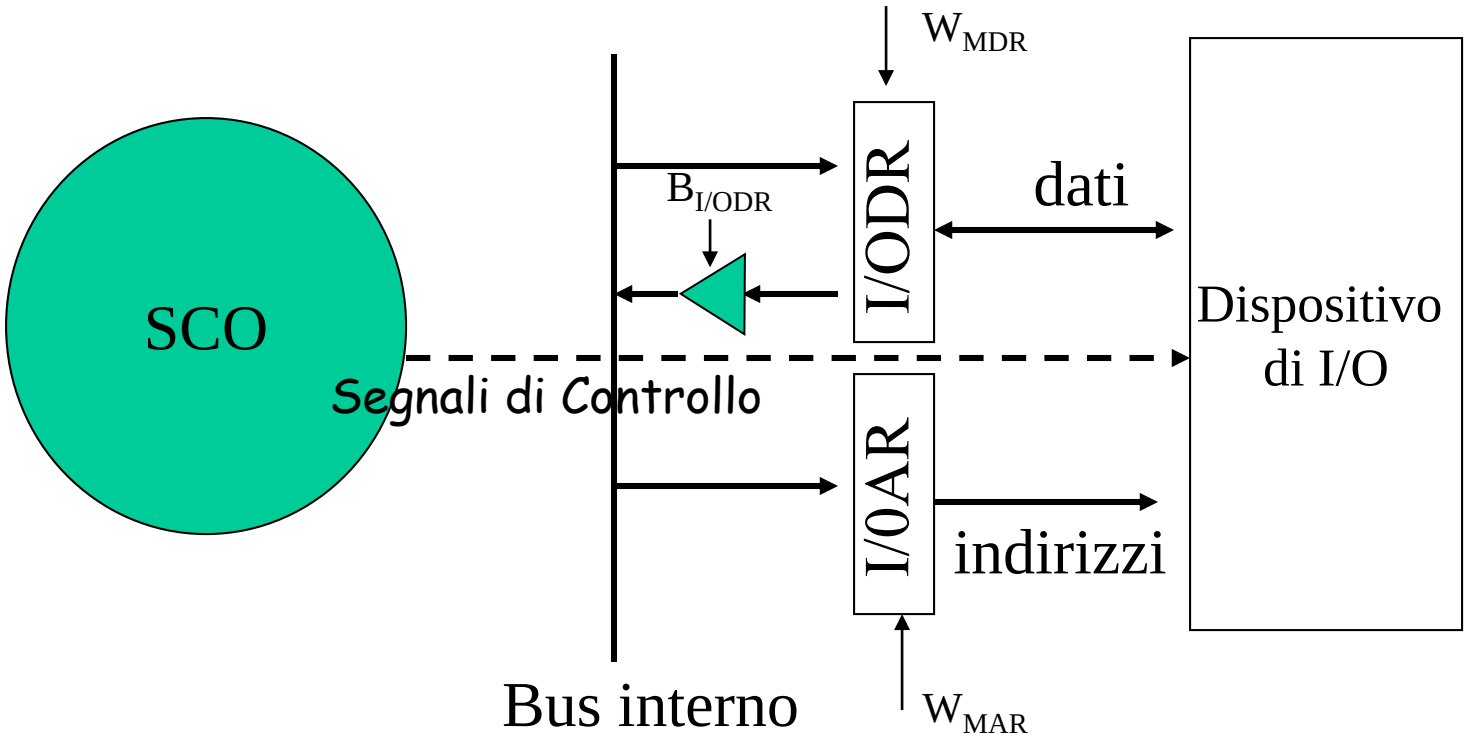
Possibili connessione tra CPU e dispositivi di Ingresso/Uscita

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

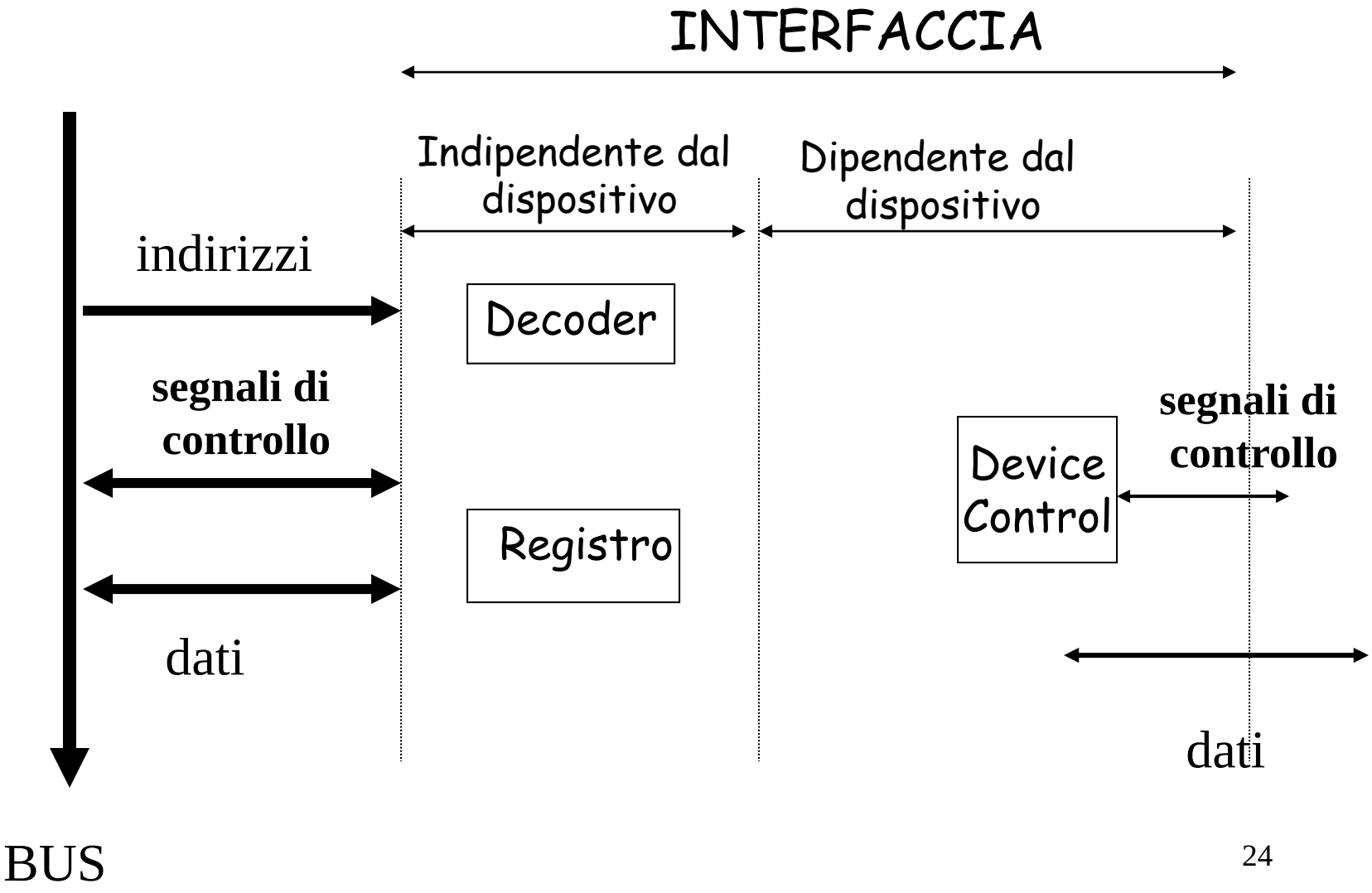


Dispositivi di I/O: interfaccia dello z64

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



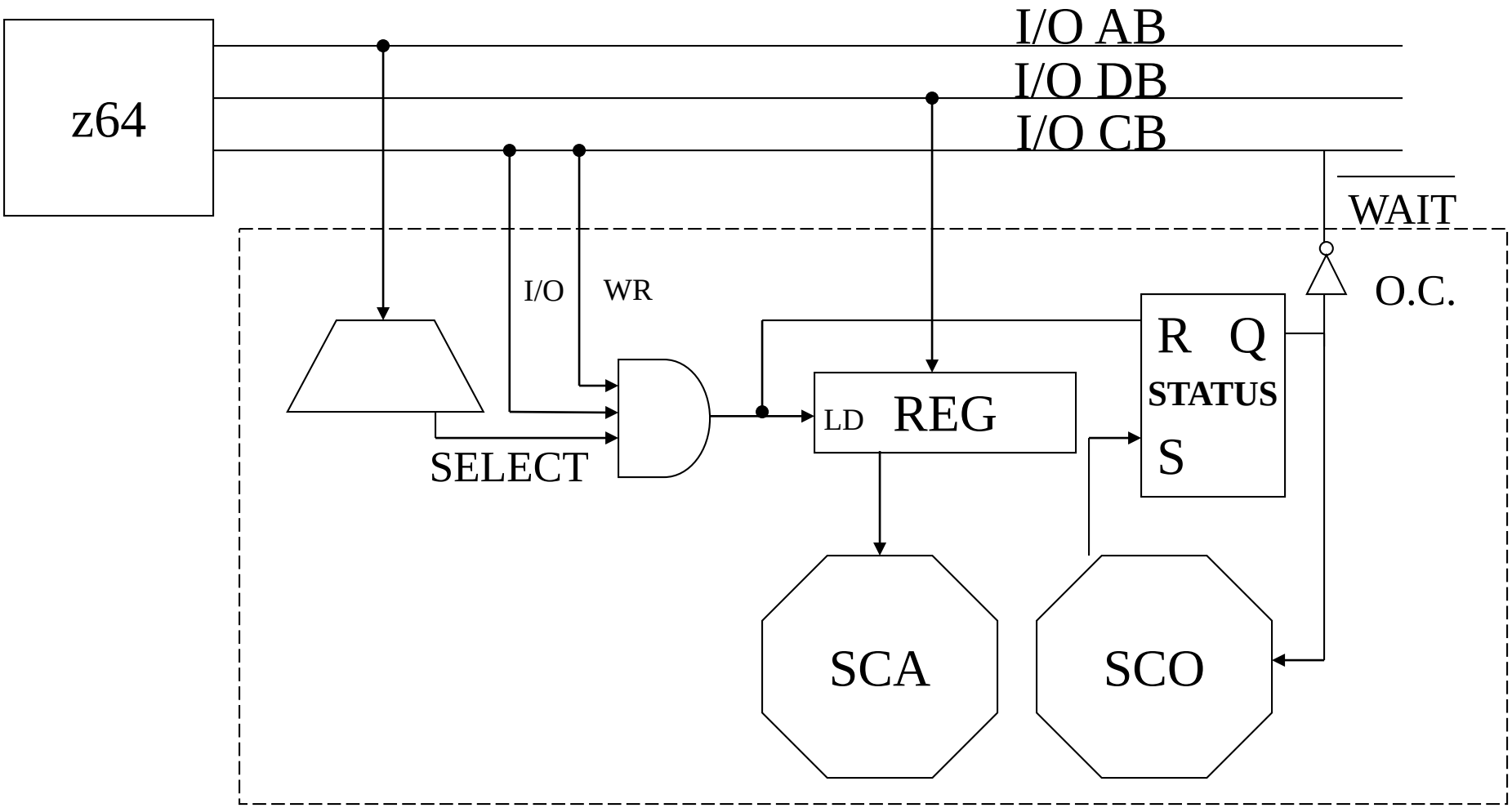
Interfaccia dispositivi di I/O



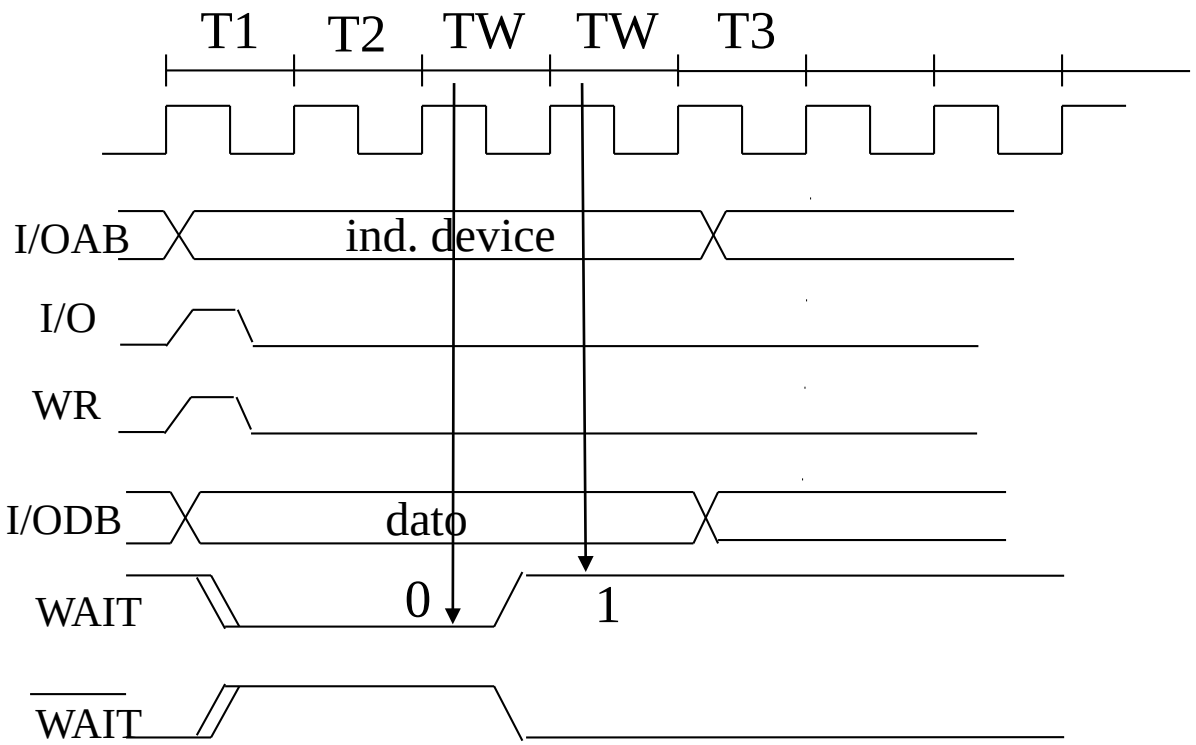
I/O programmato INTERFACCIA DISPOSITIVI DI I/O

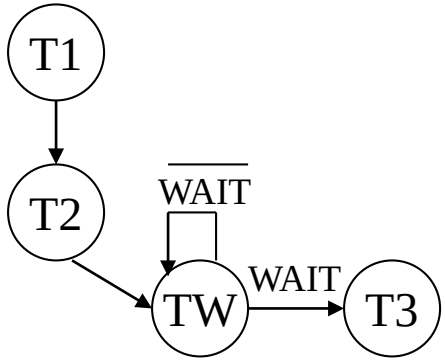
(per supportare protocollo di handshaking, implementato a firmware)

Schema di Interfaccia per l'output tra z64 e <u>più</u> DISPOSITIVI di I/O



Temporizzazione dei segnali nel caso di più dispositivi di I/O

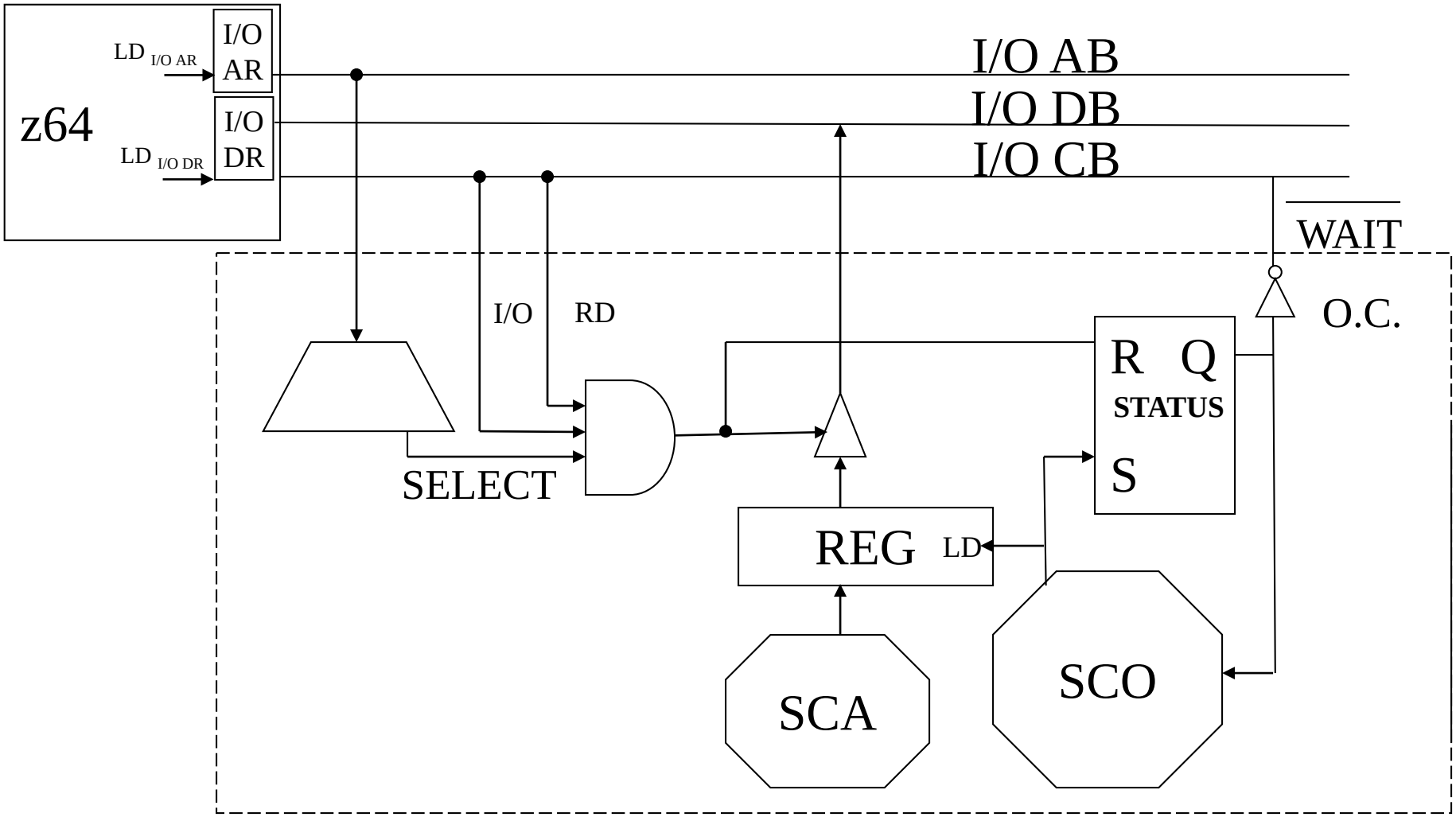




I/O programmato INTERFACCIA DISPOSITIVI DI I/O

(per supportare protocollo di handshaking, implementato a firmware)

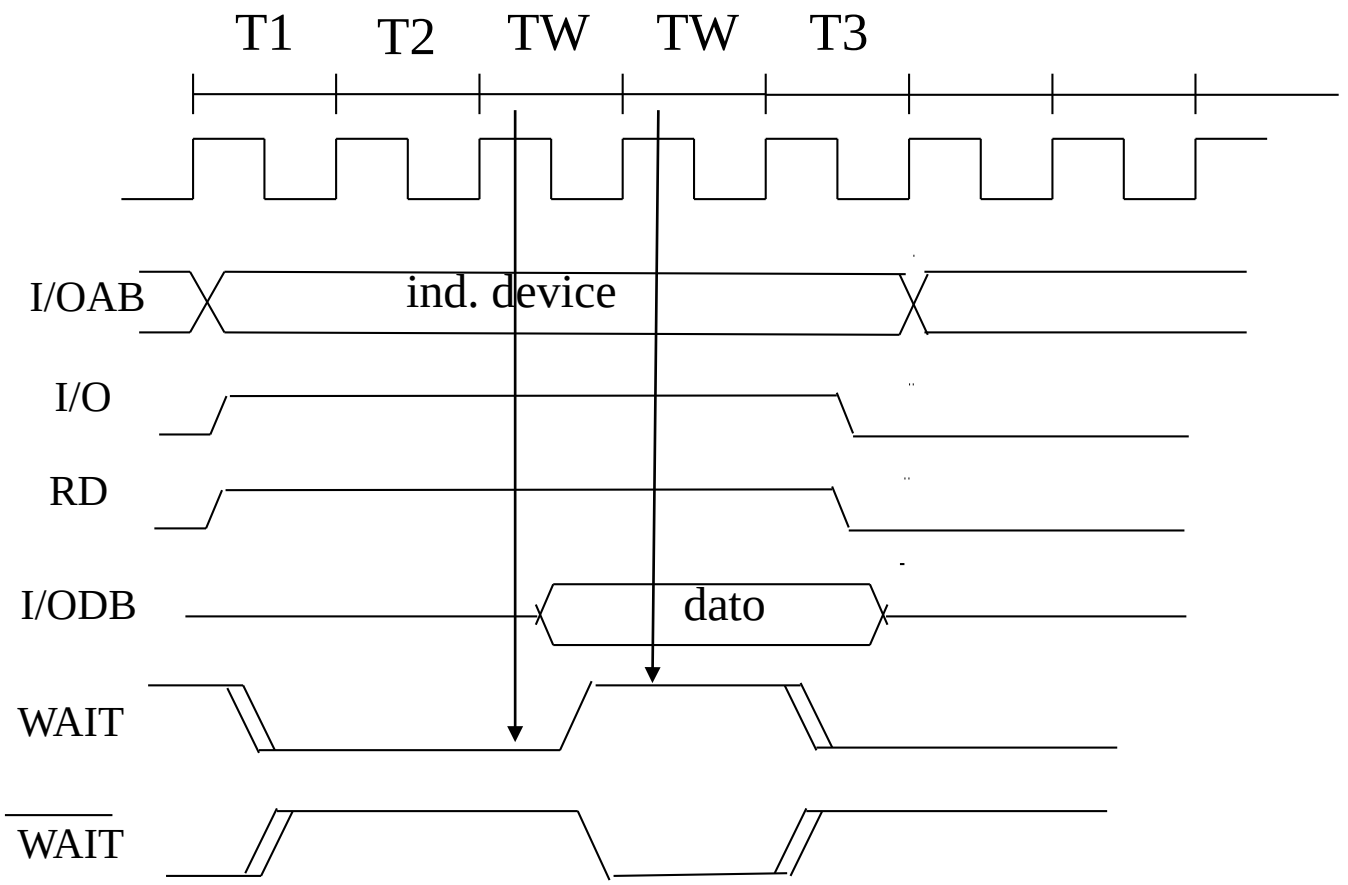
Schema di Interfaccia per l'input tra z64 e <u>più</u> DISPOSITIVI di I/O



I/O programmato INTERFACCIA DISPOSITIVI DI I/O

(per supportare protocollo di handshaking, implementato a firmware)

Schema di Interfaccia per l'input tra z64 e più DISPOSITIVI di I/O



Porte Logiche Open Collector e Connessione Wired-OR

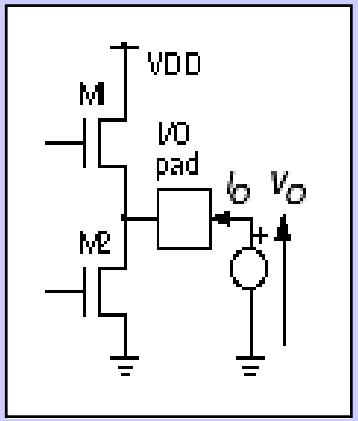
Porte Logiche: Totem Pole vs Open Collector

- 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 transistore di *pull-up attivo* che forza un livello di tensione alto sul pin d’uscita.
    - In caso di uscita logica “bassa”, un transistore 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)

Porte Logiche: Totem Pole vs Open Collector

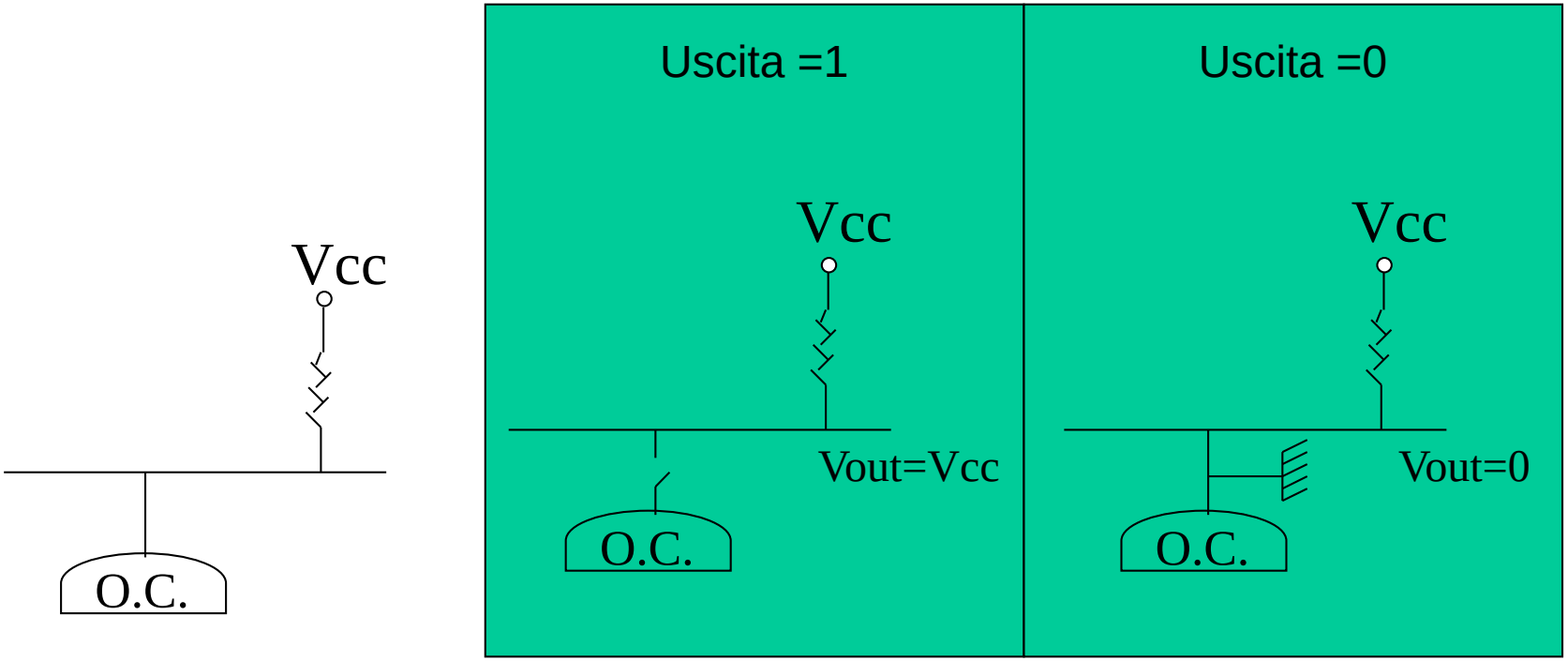
- E’ possibile distinguere due ti… logic… porte logiche in fun… d… che le implementa…
  - Po…
    - In … …vo ch…
    - In … che forz…
  - Open …
    - In ca… …a in alta impedenz…
    - In caso di uscit… …sione sul pin d’uscita vale 0 (il pin d’uscita è mess… massa)

Curiosità: Perché “Totem Pole”?



Porte Logiche Open Collector

- Poiché in configurazione open-collector le porte non possono generare autonomamente lo stato logico alto, occorre utilizzare un generatore di tensione ed una resistenza di pull-up:



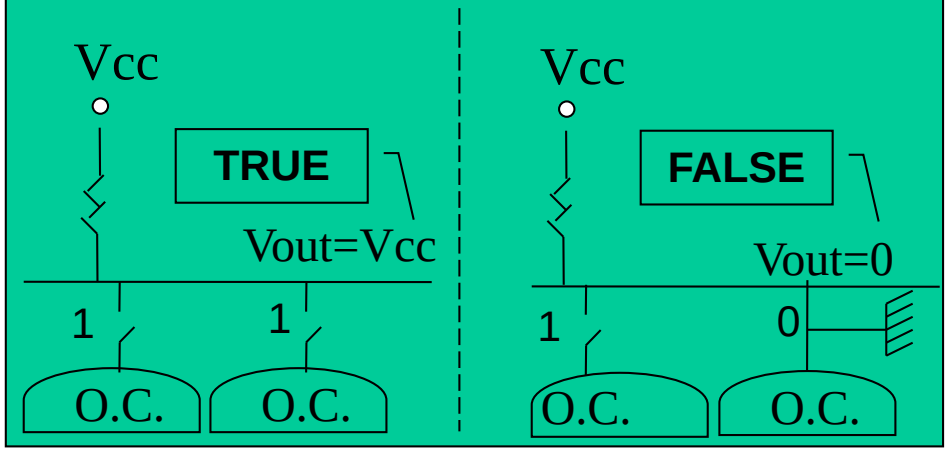
WIRED OR / WIRED AND

Connettendo su una stessa linea più porte open collector otteniamo le cosiddette connessioni WIRED OR, ovvero WIRED AND a seconda che si lavori in logica positiva o negativa:

LOGICA POSITIVA

1) Se solo una porta ha l'uscita bassa (FALSE), la linea va a massa e l'uscita è bassa (FALSE).
2) Per ottenere un'uscita alta (TRUE), tutte le porte devono avere uscita alta (TRUE).

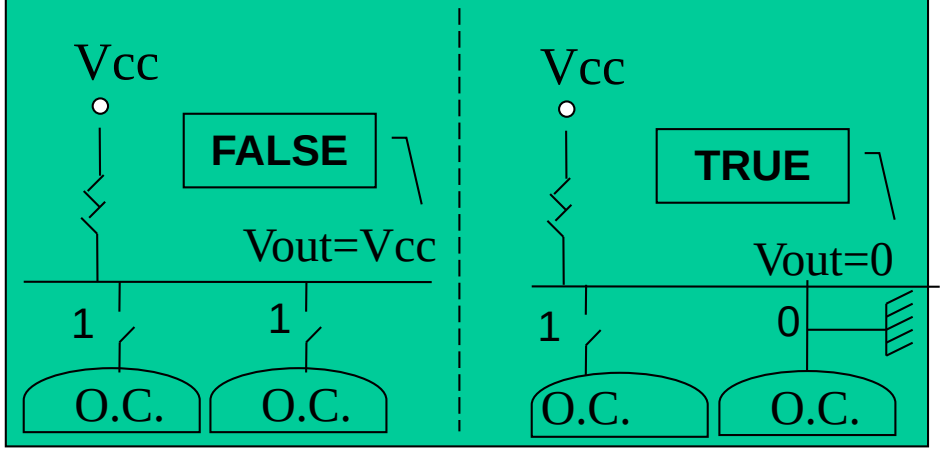
AND DELL'USCITA DELLE SINGOLE PORTE



LOGICA NEGATIVA

1) Se solo una porta ha l'uscita bassa (TRUE), la linea va a massa e l'uscita è bassa (TRUE).
2) Per ottenere un'uscita alta (FALSE), tutte le porte devono avere uscita alta(FALSE).

OR DELL'USCITA SINGOLE PORTE



Connessione di più porte logiche su una stessa linea (BUS)

- **PROBLEMA**: Non è possibile connettere più porte logiche Totem Pole su una stessa linea onde evitare conflitti dovuti alla presenza di stati logici diversi su porte logiche diverse.
- **Soluzioni**:
  - Utilizzare buffer three states oppurtunamente pilotati per garantire che solo una porta logica sia effettivamente connessa al bus in ogni istante

    (**che utilizzeremo successivamente per i dati**)
  - Utilizzare una connessione wired-or per “porta logica”

    (**che utilizzeremo successivamente per i segnali di controllo**)

Esempio: Connesione, in wired OR, di più interfacce alla linea “not READY”

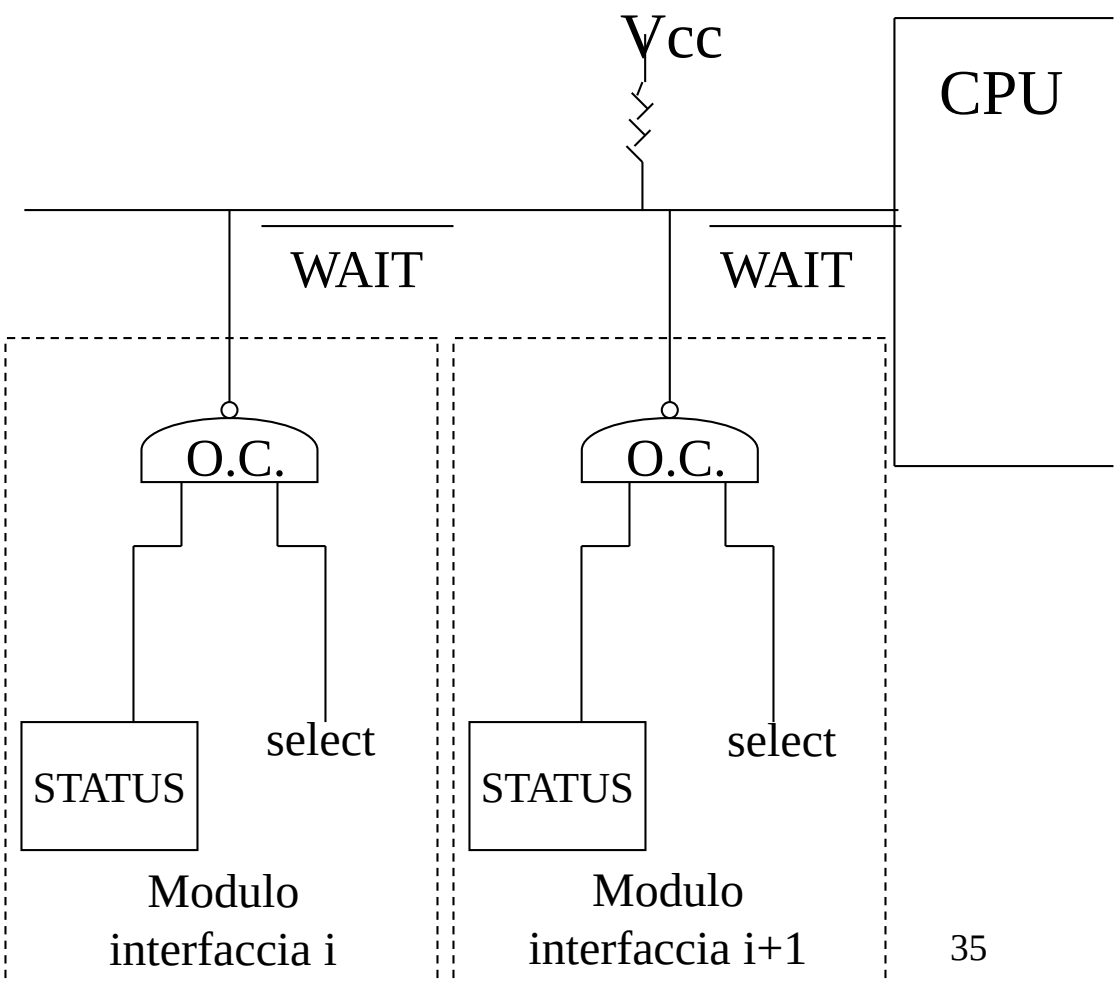
A) Se nessuna interfaccia ha attivo il segnale di select, tutti i NAND O.C. vanno in alta impedenza e $\overline{\text{WAIT}}$ =1 (false).

B) Poiché solo una interfaccia può avere il segnale di select attivo:

1) Solo tale interfaccia può avere il segnale $\overline{\text{WAIT}}$ =0 (se STATUS=1);

2) Tutte le altre interfacce avranno $\overline{\text{WAIT}}$ =1, ovvero in alta impedenza.

Questo ci consente di evitare conflitti!



I/O programmato

PROTOCOLLO DI HANDSHAKING, IMPLEMENTATO A ***FIRMWARE***

Aspetto negativo: l'attesa della produzione o del consumo del dato da parte della periferica può bloccare per lunghi periodi la CPU

Per esempio se CK del processore 1 nsec. (10^{-9} sec.) e velocità di produzione/consumo dei dati della periferica è 1 msec (10^{-3} sec.) allora il processore, per ogni interazione, "passa" nello stato di WAIT per un numero di volte dell'ordine di 10^{6} .

I/O programmato
INTERFACCIA DISPOSITIVI DI I/O
(PER SUPPORTARE PROTOCOLLO DI HANDSHAKING, IMPLEMENTATO A ***SOFTWARE***)

1) necessità di avvertire la periferica che il processore ha scritto o vorrebbe leggere un dato dal registro di interfaccia della periferica

2) necessità di verificare che il dispositivo abbia letto o prodotto un dato, che negli schemi precedenti era verificato utilizzando la variabile di condizione WAIT (negata)

Periferica che avrà un'interfaccia molto simile a quella in cui c'era il segnale WAIT (negato)

I/O programmato
INTERFACCIA DISPOSITIVI DI I/O

(PER SUPPORTARE PROTOCOLLO DI HANDSHAKING, IMPLEMENTATO A ***SOFTWARE***)

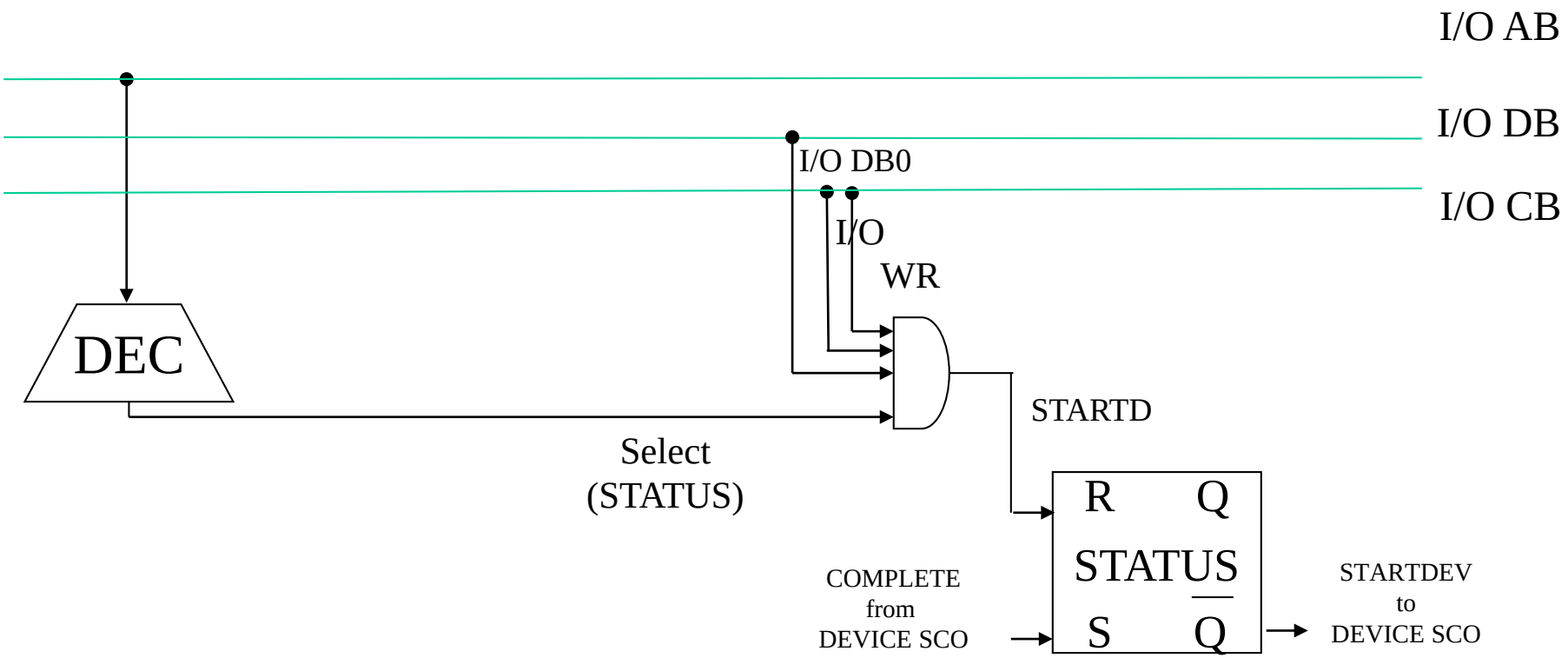
1) Per avvertire la periferica che il processore ha scritto o vorrebbe leggere un dato dal registro di interfaccia della periferica si utilizzerà sempre il flip/flop di STATUS che verrà resettato da processore tramite una istruzione di OUTPUT, pertanto anche in questo caso il flip/flop di handshaking STATUS verrà visto dal processore come un registro (di un solo bit)

2) Per verificare che il dispositivo abbia letto o prodotto un dato si utilizzerà sempre il flip/flop di STATUS, solo che questa volta la sua uscita verrà letta dal processore tramite una istruzione di INPUT per verificare poi se tale uscita abbia il valore 0 o il valore 1. In tal caso il flip/flop di handshaking STATUS verrà visto dal processore come un registro (di un solo bit) e pertanto dovrà avere un indirizzo differente da quello del registro di interfaccia dove vengono trasferiti i dati

I/O programmato – INTERFACCIA di INPUT

PROTOCOLLO DI HANDSHAKING IMPLEMENTATO A SOFTWARE

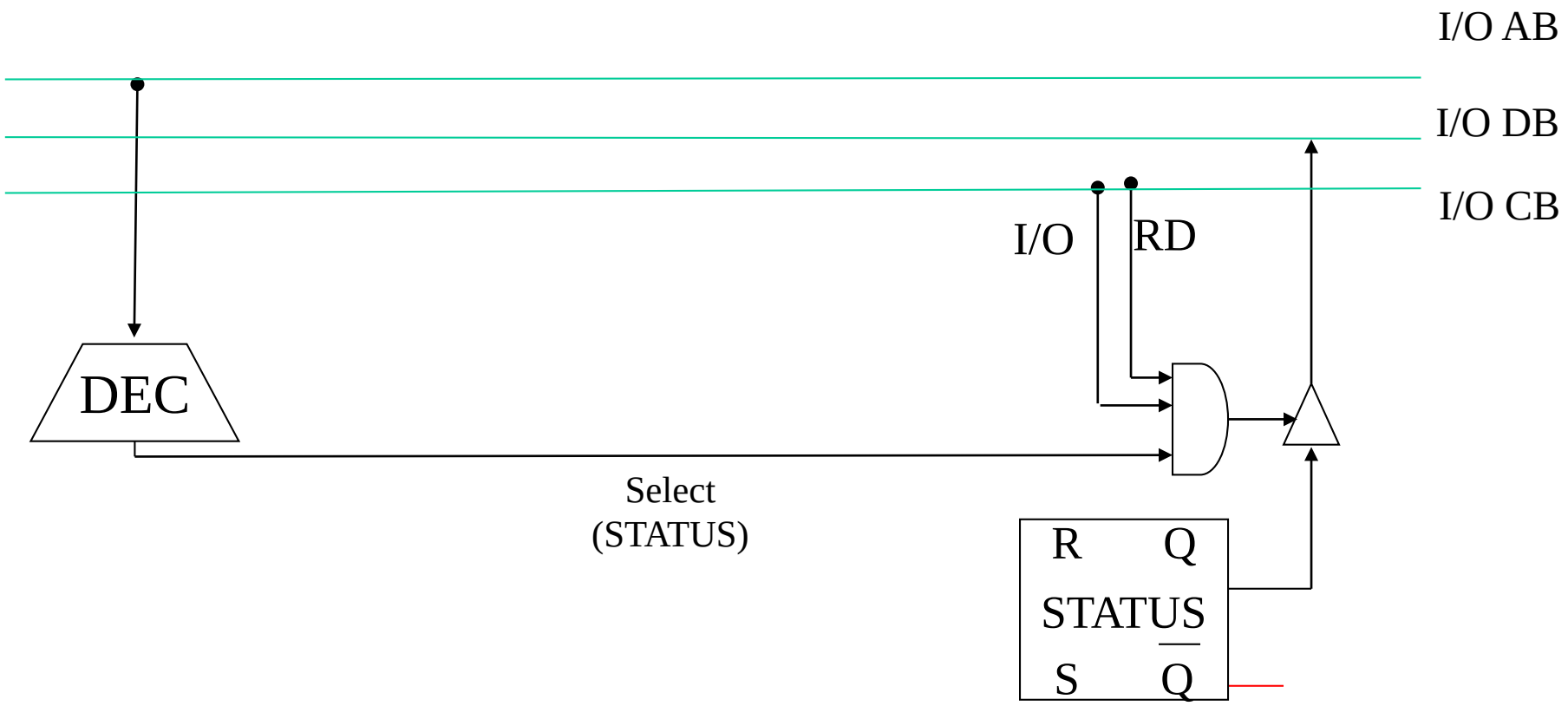
Hardware necessario per avvertire la periferica

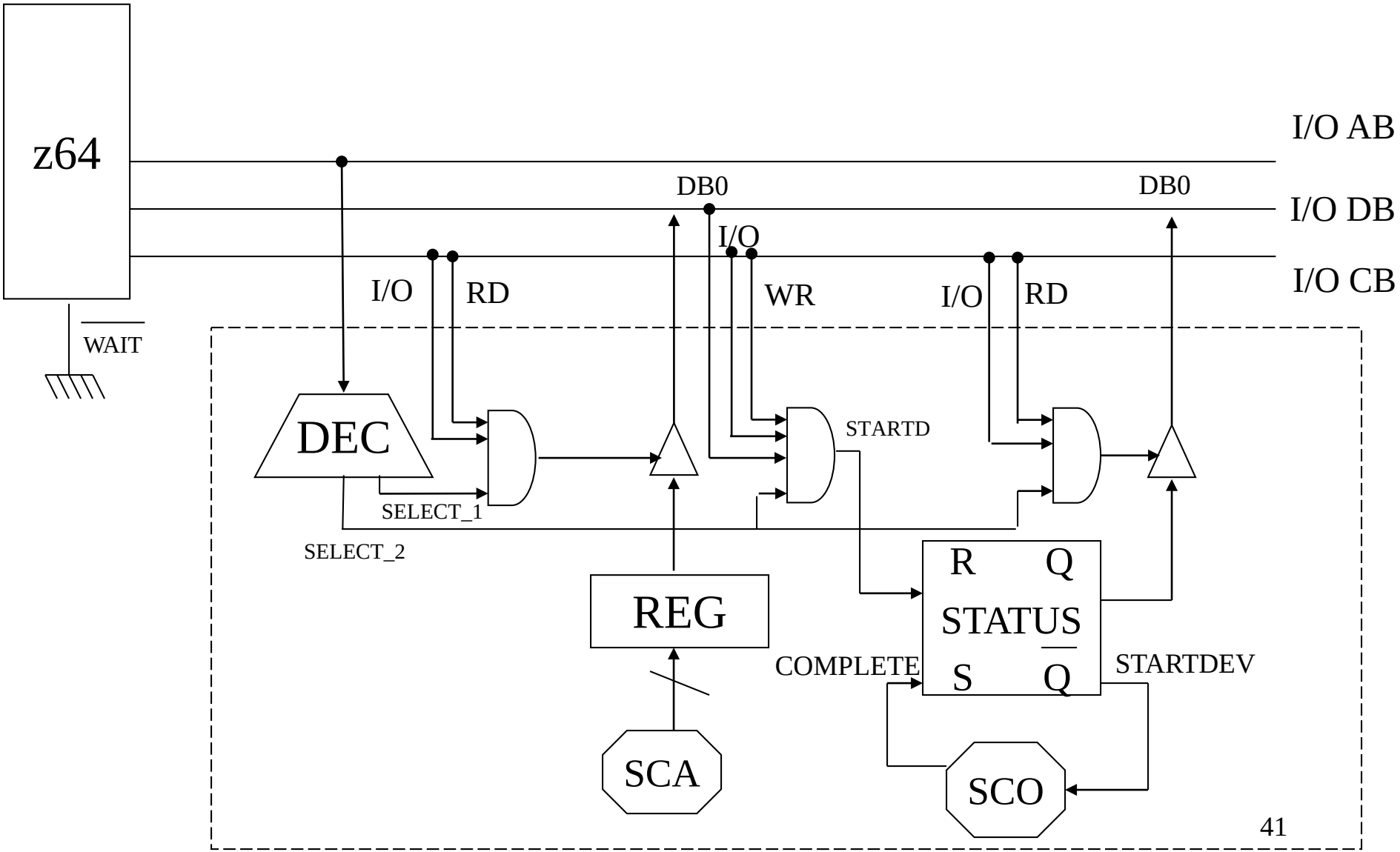


I/O programmato – INTERFACCIA di INPUT

PROTOCOLLO DI HANDSHAKING IMPLEMENTATO A SOFTWARE

Hardware necessario per verificare se la periferica è pronta




z64
WAIT
I/O AB
I/O DB
I/O CB
DB0
DB0
I/O
I/O
RD
WR
I/O
RD
DEC
STARTD
SELECT_1
SELECT_2
REG
R
Q
STATUS
S
Q
COMPLETE
STARTDEV
SCA
SCO
41

I/O programmato – INTERFACCIA di INPUT

PROTOCOLLO DI BUSY WAITING IMPLEMENTATO A SOFTWARE

1. Il processore avverte il dispositivo che vuole un dato da lui (resettando il flip/flop STATUS).
2. Il processore verifica che il dispositivo abbia prodotto il dato, ciò lo verifica testando il flip/flop STATUS.
3. Se il valore di STATUS è 0 il processore deve attendere e pertanto ritorna al punto 2.
4. Se il valore di STATUS è 1 il processore esegue una istruzione di INPUT (seleziona il dispositivo ed invia il segnale di controllo IO RD per trasferire il dato presente in REG all'interno di uno dei registri del processore).

Porzione di programma assembly per lettura di un dato in busy waiting

```
                movw $FF_STATUS, %dx
                movb $1, %al
                outb %al, %dx          # avverti la periferica
.aspetta:       inb %dx, %al
                btb $0, %al            # copia il bit n.0 del reg.
                                        # A nel carry
                jnc .aspetta           # attendi per. sia pronta
                movw $DEVICE_IN, %dx
                inl %dx, %eax          # input da periferica
```

Programma assembler (input)

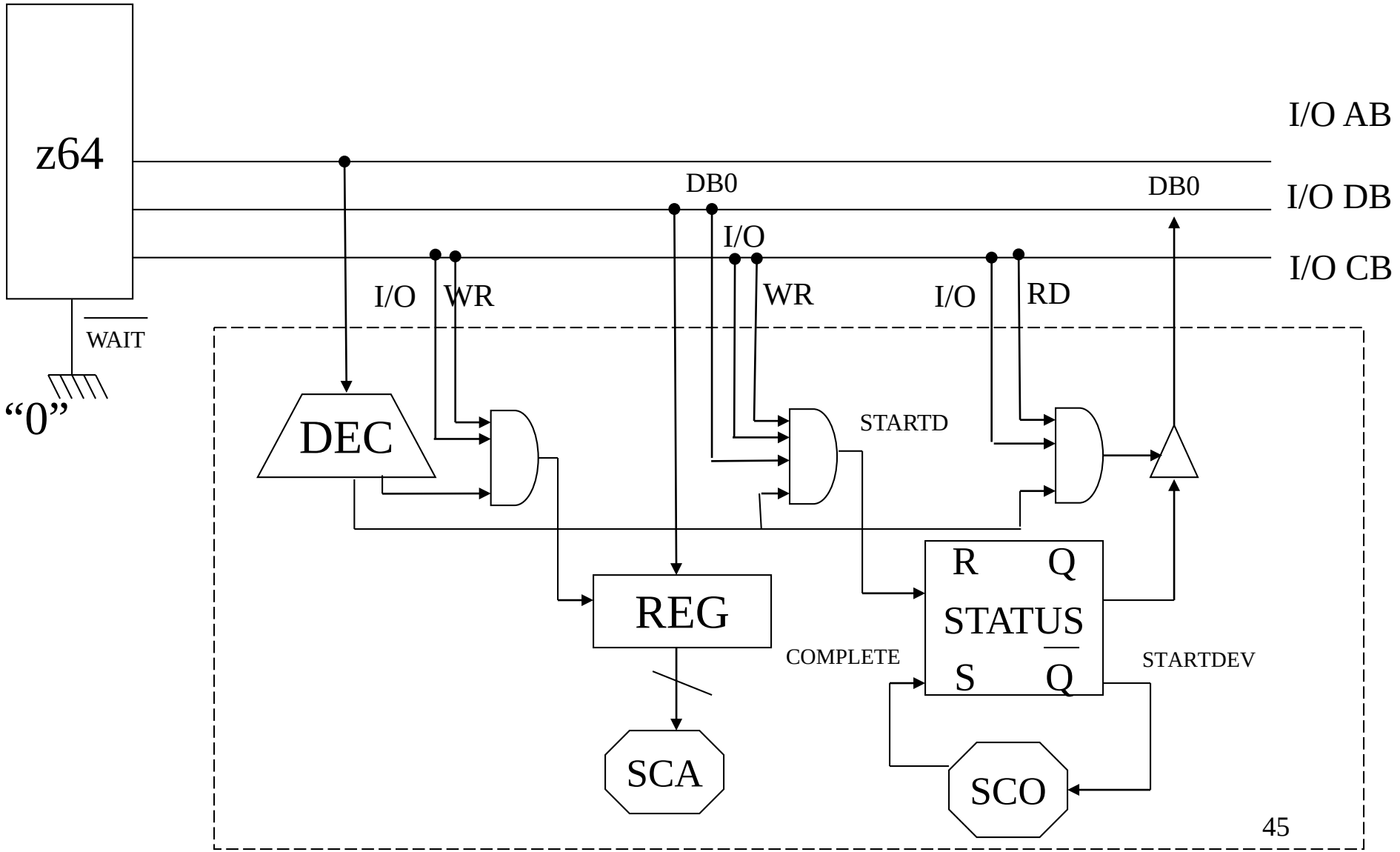
(indirizzo del registro del device memorizzato in %dx)

L’istruzione di input utilizzata in questo esempio prevede che i dati acquisiti siano di una long word. In caso di dati di dimensioni differenti, le altre varianti sono:

```
inb %dx, %al
inw %dx, %ax
inq %dx, %rax
```

I/O programmato- INTERFACCIA di OUTPUT

PROTOCOLLO DI HANDSHAKING IMPLEMENTATO A SOFTWARE



Protocollo di Output

1. Il processore esegue una istruzione di OUTPUT e trasferisce il contenuto di un registro nel registro di interfaccia del dispositivo (mediante il segnale di controllo I/OWR).
2. Il processore avverte il dispositivo che vuole un dato da lui (resettando il flip/flop STATUS).
3. Il processore verifica che il dispositivo abbia prodotto il dato, ciò lo verifica testando il flip/flop STATUS.
4. Se il valore di STATUS è 0 il processore deve attendere e pertanto ritorna al punto 2.
5. Se il valore di STATUS è 1 il processore esce dallo stato di attesa e può proseguire le sue attività

Protocollo di Output

HP: la periferica è ad uso esclusivo del processore e il processore attende in busy waiting il consumo del dato che ha trasmesso alla periferica

1. il processore esegue una istruzione di OUTPUT e trasferisce il contenuto di un registro nel registro di interfaccia del dispositivo (mediante una istruzione di output che genera i segnali di controllo I/O e WR, mette l'indirizzo della porta sull'AB e il valore da scrivere sul DB).
2. Il processore avverte il dispositivo che gli ha trasferito un dato. Ciò viene fatto resettando il flip-flop STATUS e il flip/flop rimane in tale stato per tutta la durata delle operazioni di consumo del dato da parte del dispositivo. Il reset del flip/flop viene effettuato mediante una istruzione di output che genera i segnali di controllo I/O e WR, mette ad 1 la linea meno significativa del bu dati e mette l'indirizzo della porta sull'AB,. Quando il dato è stato letto dallo SCO della periferica dal registro di interfaccia (REG), il dispositivo genera il segnale COMPLETE, settando il flip/flop STATUS.
3. Nel frattempo il processore, in attesa, può esaminare lo stato del flip/flop campionandone l'uscita, ciò viene fatto leggendone l'uscita (mediante una istruzione di input che genera i segnali di controllo I/O, RD e mette l'indirizzo della porta sull'address bus) e verificandone, con altre istruzioni che non interessano la periferica, il valore del carry (JC).
4. Se CARRY= 0 il processore deve attendere ed eventualmente tornare al punto 3.
5. Se CARRY= 1 il processore può eseguire un'altra istruzione.

Porzione di programma assembly handshaking per un dato

```
                movl $dato, %eax
                movw $DeviceOUT, %dx
                outl %eax, %dx                  # out

                movw $FF_STATUS, %dx
                movb $1, %al
                outb %al, %dx                   # start

.aspetta:       inb %dx, %al                    # Ciclo di Busy Waiting
                btb $0, %al                     # Copia bit #0 nel Carry…
                jnc .aspetta                    #... se è 0, ``aspetta″
```

Programma assembler (output)

(indirizzo del registro del device memorizzato in %dx)

L’istruzione di output utilizzata in questo esempio prevede che i dati acquisiti siano di un byte. In caso di dati di dimensioni differenti, le altre varianti sono (come per le istruzioni di movimento dati):

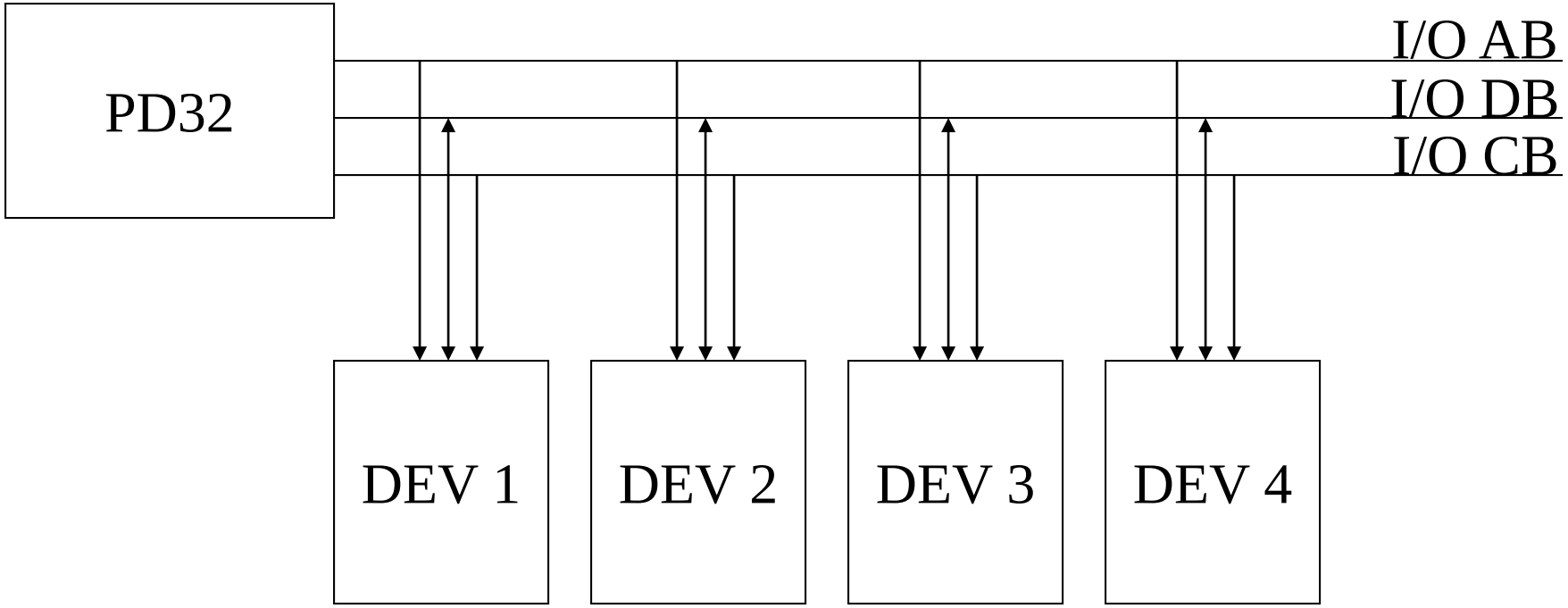
outw %ax, %dx
outl %eax, %dx
outq %rax, %dx

I/O programmato
MODALITA' BUSY WAITING
acquisizione di 100 dati

```
…
   movl $0, %ecx                    # contatore dei dati inizializzato a 0
   movq $DATI, %rdi                 # memorizzazione della locazione di memoria
                                    #  iniziale del vettore dove memorizzare i dati da
                                    # acquisire in rdi
.loop:
   movw $AD_STATUS, %dx
   movb $1, %al
   outb %al, %dx                    # avvia (start) la periferica a produrre dati
.bw:
   inb %dx, %al
   btb $0, %al
   jnc .bw                          # attendo che la periferica sia pronta
   movw $AD_REG, %dx                # inizializzazione del registro di porta di I/O
   inw %dx, %ax                     # prelevo il dato dalla periferica…
   movw %ax, (%rdi, %ecx, 2)        # …lo copio in memoria (nel vettore)
   addl $1, %ecx                    # incremento il contatore
   cmpl $100, %ecx
   jnz .loop
   hlt
```

Polling

(verifica circolare se ogni DEVICE è pronto ad interagire)



Polling (1/2)

(esempio di acquisizione di 100 word da due periferiche)

```
  movl $0, %ecx          # contatore dei dati inizializzato a 0
  movb $1, %al           # valore ‘1’ per avviare le periferiche
  movw $AD1, %dx
  outb %al, %dx          # Avvia AD1
  movw $AD2, %dx
  outb %al, %dx          # Avvia AD2

.poll:
  movw $AD1, %dx
  inb %dx, %ax           # leggo il valore di STATUS da AD1
  btb $0, %ax
  jc .acquisisci_dati1           # se è pronta acquisisco dati
  movw $AD2, %dx
  inb %dx, %ax           # leggo il valore di STATUS da AD2
  btb $0, %ax
  jc .acquisisci_dati2           # se è pronta acquisisco dati
  jmp .poll                      # proseguo con il ciclo di polling
```

Polling (2/2)

(continua esempio di acquisizione di 100 word da due periferiche)

```
.acquisisci_dati1:
   movw $AD1_REG, %dx
   call acquisisci                  # chiama la routine di acquisizione
   movb $1, %al                     # valore '1' per avviare le periferiche
   movw $AD1, %dx
   outb %al, %dx                    # riavvia AD1
   jmp .poll                                # proseguo con il ciclo di polling

.acquisisci_dati2:
   movw $AD2_REG, %dx
   call acquisisci                  # chiama la routine di acquisizione
   movb $1, %al                     # valore '1' per avviare le periferiche
   movw $AD2, %dx
   outb %al, %dx                    # riavvia AD2
   jmp .poll                                # proseguo con il ciclo di polling

acquisisci:
   inw %dx, %ax                     # prelievo del dato e…
   movw %ax, vettore(, %ecx, 2) # … suo trasferimento in memoria
   addl $1, %ecx                    # incremento del contatore
   cmpl $100, %ecx                  # controllo di uscita dal ciclo di polling
   jnz .return
            hlt
.return:
   ret
```

Trasferimenti di stringhe in busy waiting cablato

InsX

(String Input from I/O Port)

Trasferisce un numero di dati (la dimensione di un singolo elemento è specificato dalla X, che può essere B, W, L o Q) pari al valore memorizzato nel registro %rcx dalla porta di ingresso identificata dal contenuto del registro %dx nelle locazioni di memoria il cui indirizzo iniziale è memorizzato nel registro %rdi.

OutsX
(String Output to I/O Port)

Trasferisci un numero di dati (la dimensione di un singolo elemento è specificato dalla X, che può essere B, W, L o Q) pari al valore memorizzato nel registro %rcx dalle locazioni di memoria il cui indirizzo iniziale è memorizzato nel registro %rdi alla porta di uscita identificata dal contenuto del registro %dx.

Trasferimenti di stringhe in busy waiting cablato

Vantaggi:

- programmi più corti
- risparmio delle fasi di fetch

Implementazione dei trasferimenti di stringhe in busy waiting cablato

Le attività del SCO sono simili a quelle effettuate per l'esecuzione di un programma assembly in cui è prevista l'acquisizione dei dati in busy waiting, quindi per esempio nel caso di implementazione della InsB

- Avvertire la periferica che desidera un dato
- Attendere che il dato sia pronto
- Acquisire il dato (equivalente a IN)
- Mettere nella locazione di memoria il cui indirizzo è dato dal valore del registro %rdi (equivalente a MOV)
- Incrementare di 1 (essendo il dato un Byte) il valore del registro %rdi (equivalente a ADD)
- Decrementare il contenuto del registro %rcs (equivalente a SUB)
- Verificare se il contenuto del registro %rcs è pari a zero
- Se pari a zero effettuare il fetch dell'istruzione successiva, se pari ad uno ritorna alla prima attività (Avvertire la periferica che desidera un dato)

I/O programmato

- SVANTAGGI –

- Uso non efficiente del processore (perdita di tempo per verificare se la periferica è pronta o meno ad interagire);

- Rischio di non soddisfare esigenze di urgenza (real-time) (p.e. nel polling vengono visitate le periferiche in modo ciclico e quindi non si possono gestire eventi all'atto del loro verificarsi, quali caduta della tensione)

Necessità di interazione basata sulla richiesta dei dispositivi esterni (INTERRUZIONI)

INTERRUZIONE

Similitudine con la ricezione di una telefonata:

Normale interazione:
- squillo;
- prelievo cornetta;
- "pronto";
- il chiamante si identifica
- inizio del colloquio

Polling:
- squillo;
- prelievo cornetta
- richiesta di identificazione: "sei Giovanni?"
- se affermativo inizio colloquio con Giovanni
- se no, nuova richiesta di identificazione: "sei Franco?“
- se affermativo inizio colloquio con Franco
- se no, nuova richiesta…….
- ……

Naturalmente al momento della ricezione della telefonata si stava facendo qualcosa di altro (p.e. doccia), che dovrà essere continuato dopo che si è terminata la telefonata.

Inoltre il chiamante ci può "interrompere" solo se lo desideriamo, per esempio spengo il telefonino del lavoro quando sono a casa.

Fasi per la gestione dell'interruzione

a) salvare lo stato del processo in esecuzione;

b) identificare il programma di servizio relativo all'interruzione;

c) eseguire il programma di servizio;

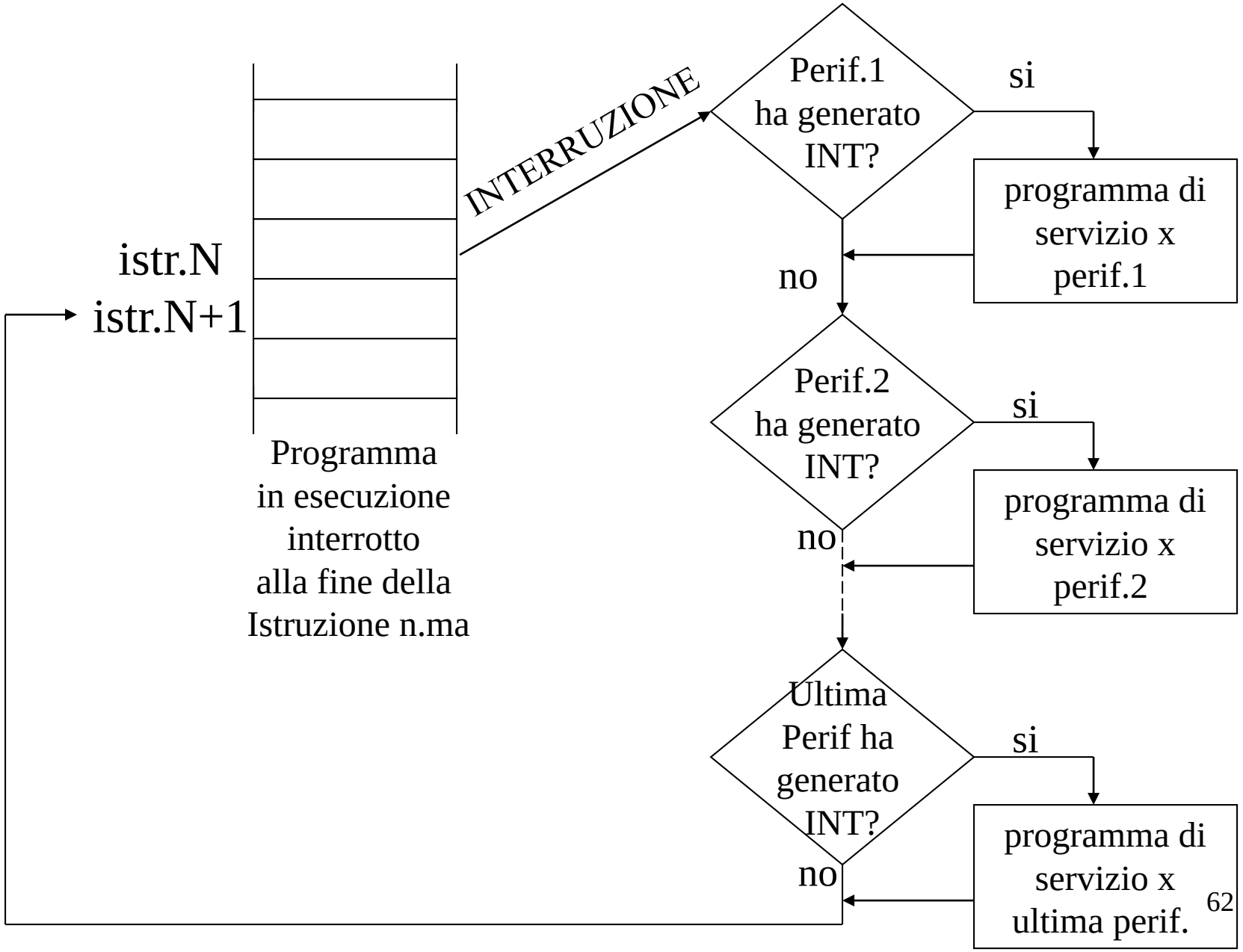
d) riprendere le attività lasciate in sospeso.

Tecniche di identificazione del programma di servizio relativo all'interruzione

Tecniche di identificazione:

- Polling
- Polling a multilivello
- Vettorizzata
- Vettorizzata a multilivello

Polling


istr.N
istr.N+1
INTERRUZIONE
Programma
in esecuzione
interrotto
alla fine della
Istruzione n.ma
Perif.1
ha generato
INT?
si
programma di
servizio x
perif.1
no
Perif.2
ha generato
INT?
si
programma di
servizio x
perif.2
no
Ultima
Perif ha
generato
INT?
si
programma di
servizio x
ultima perif.
no

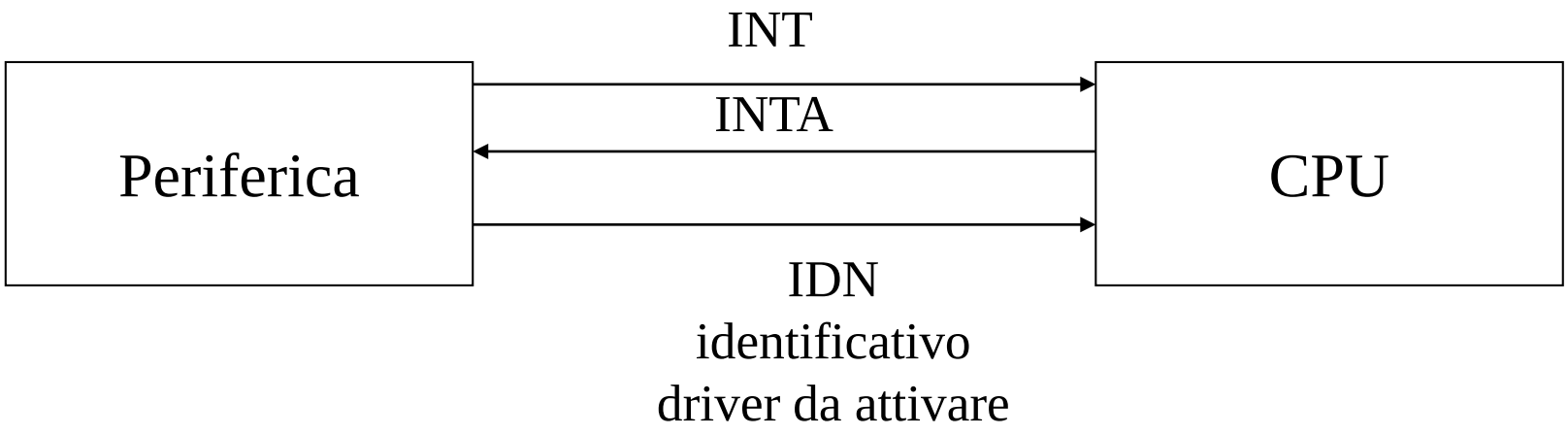
Routine di Polling per il riconoscimento delle interruzioni

Routine: JP $DISP_1$, $DRIVER_1$
JP $DISP_2$, $DRIVER_2$
….
JP $DISP_N$, $DRIVER_N$

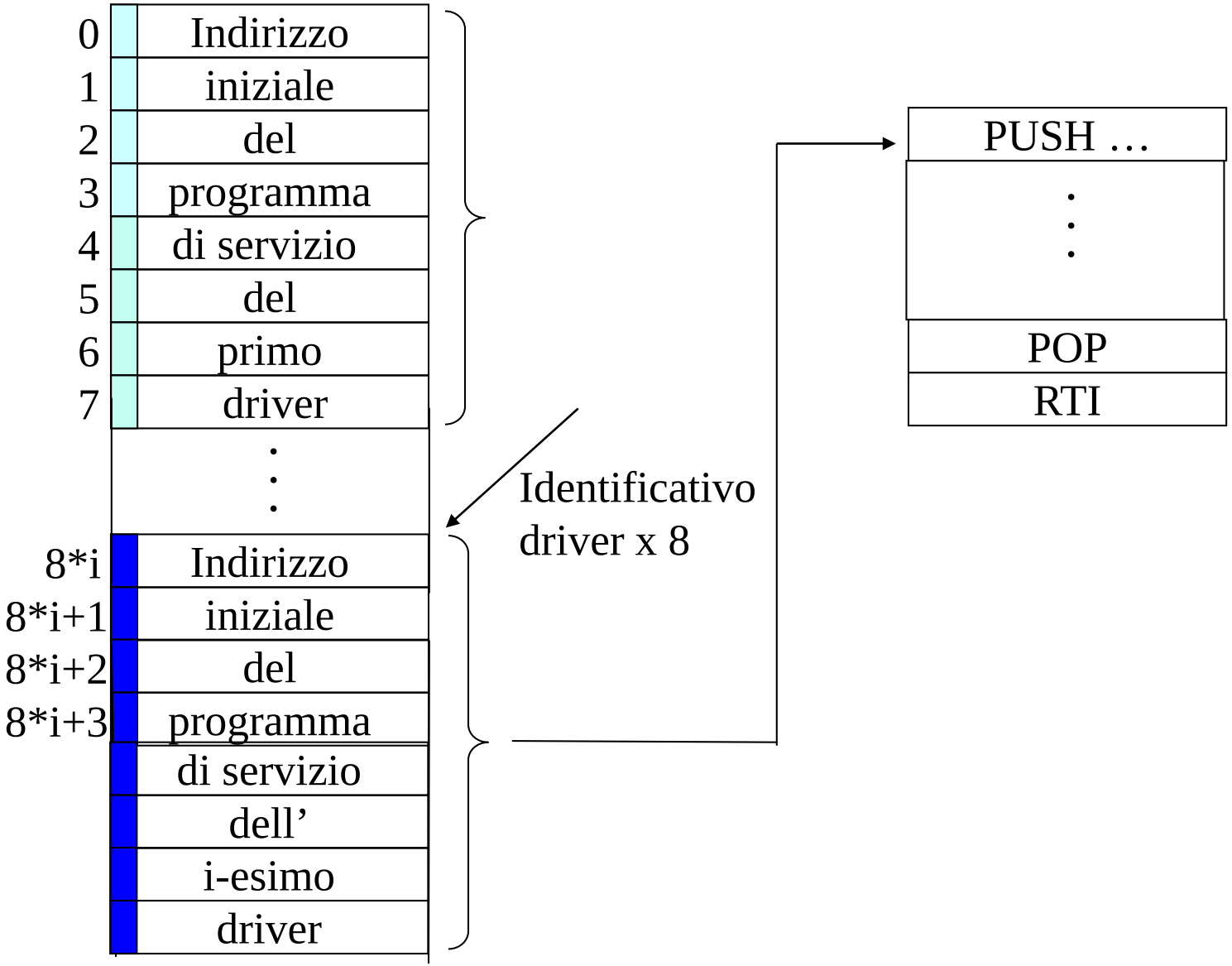
- Richieste multiple vengono servite in ordine di interrogazione
- Tempo di CPU non minimo per il riconoscimento

N.B “JP” non è una istruzione z64

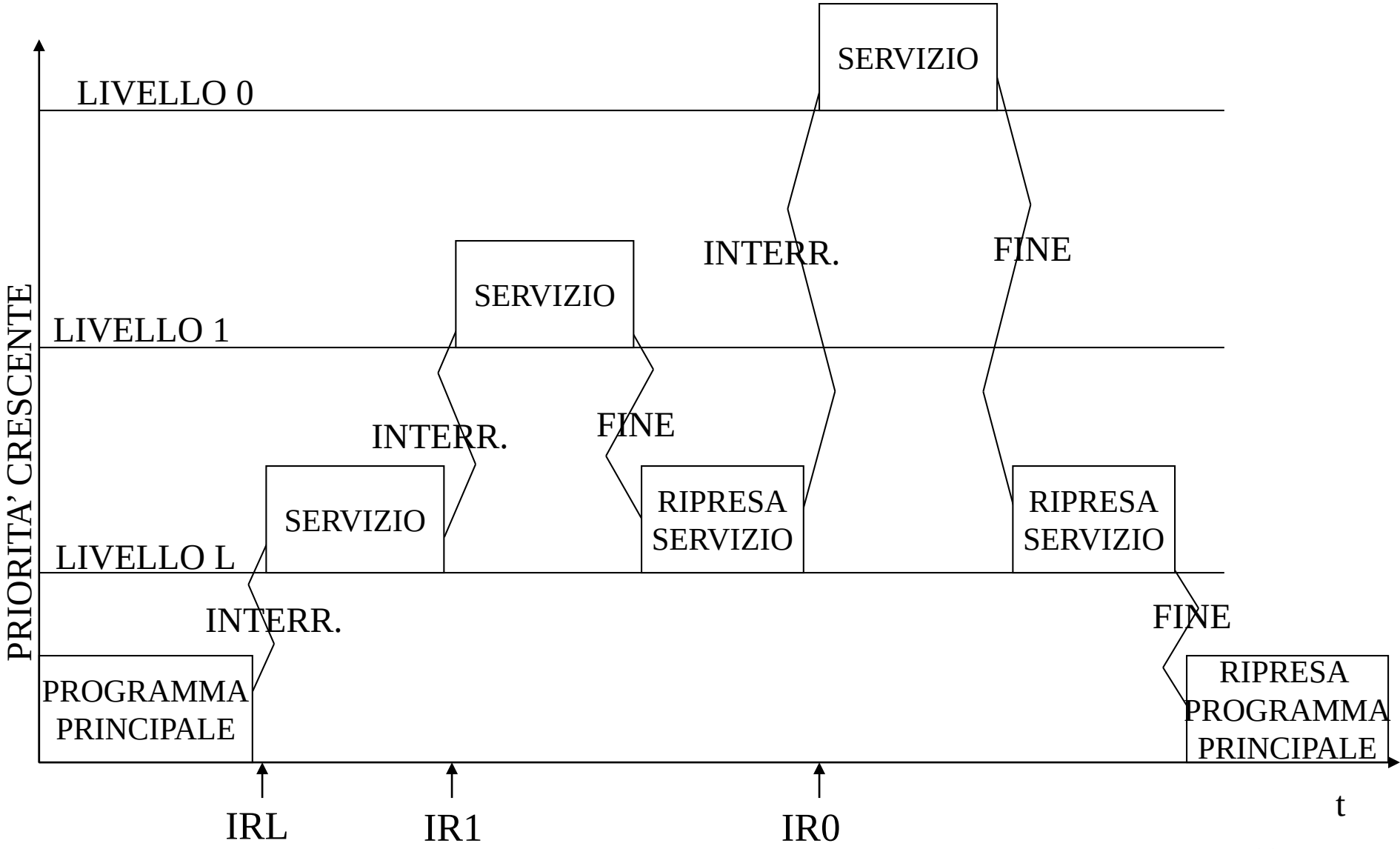
Meccanismo interruzioni vettorizzate



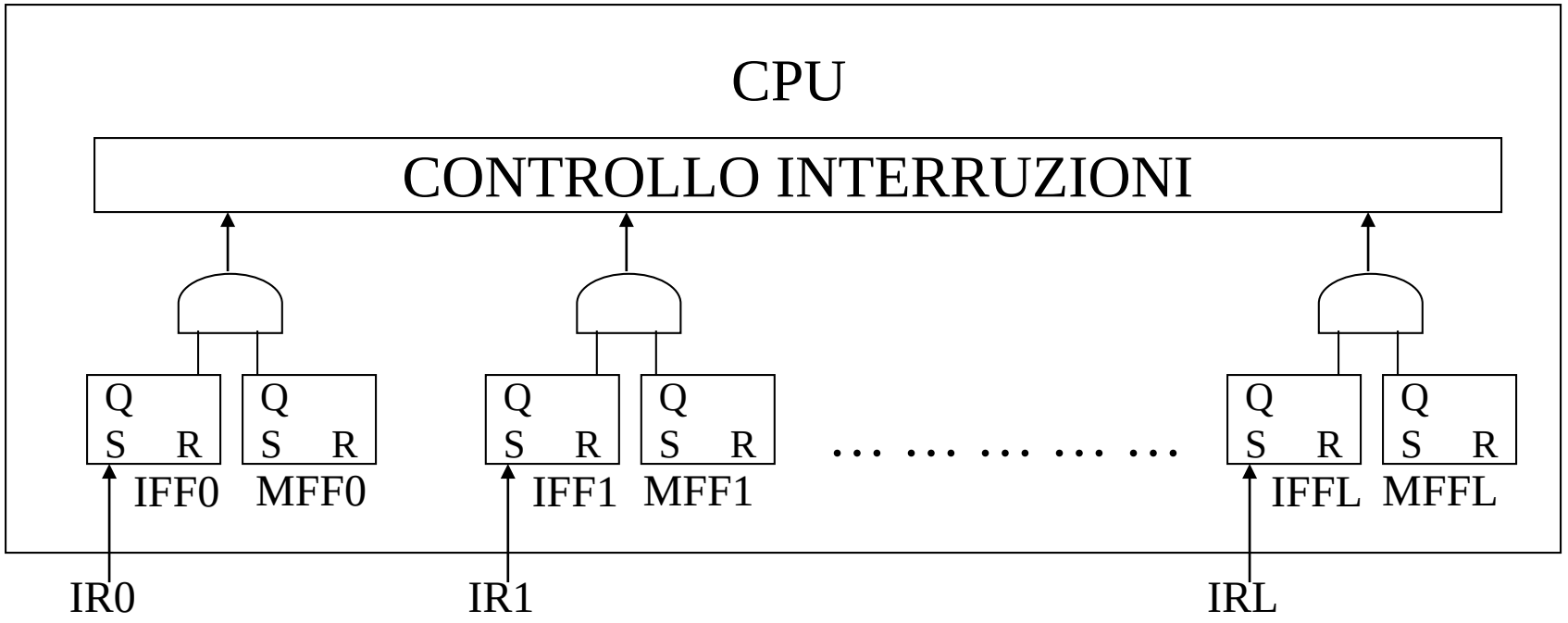
Interrupt Descriptor Table (IDT)



Meccanismo di interruzione multilivello



Meccanismo di interruzione multilivello (a polling o vettorizzata)



IFFi: Flip-Flop di memorizzazione di richiesta a livello i

MFFi: Flip-Flop di mascheramento delle richieste a livello i

(potrebbe non essere utilizzato)

Gestione delle richieste di interruzione

Abilitazione/disabilitazione delle interruzioni

Per memorizzare l'informazione che le interruzioni siano o meno abilitate si fa uso di un flip-flop (denominato *I* e contenuto nel registro SR).

Il contenuto di questo flip-flop può essere manipolato dalle istruzioni assembly CLI e STI.

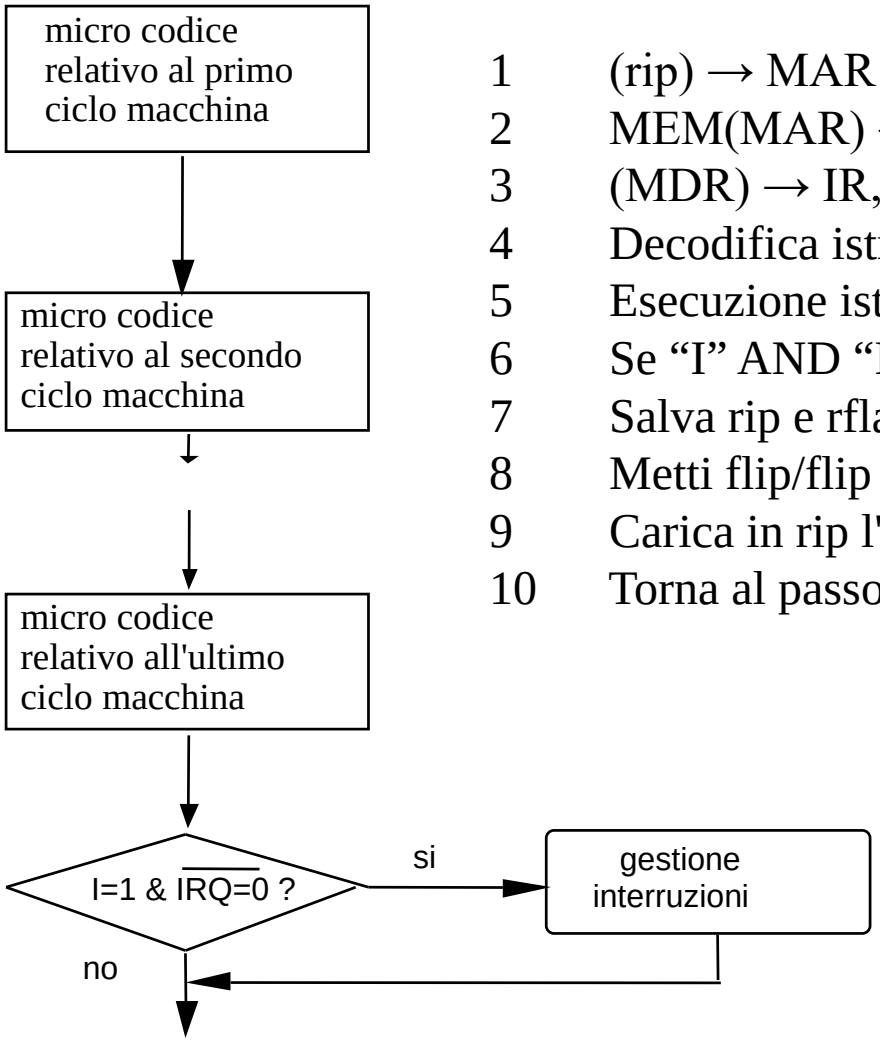
Verifica richiesta delle interruzioni

La richiesta di un'interruzione avviene in modo asincrono rispetto alle attività del processore e quindi del suo SCO.

La verifica della presenza della richiesta di un’interruzione viene fatta alla fine di ogni ciclo istruzione

Si potrebbe anche effettuarlo alla fine di ogni ciclo macchina, ma in questo caso sarebbe necessario salvare lo stato del microprogramma (più complesso da implementare).

Modifica SCO z64



1	(rip) → MAR	--- fetch
2	MEM(MAR) → MDR	--- fetch
3	(MDR) → IR, (rip)+8 → rip	--- fetch
4	Decodifica istruzione	
5	Esecuzione istruzione	
6	Se “I” AND “$\overline{\text{IRQ}}$” = 0 allora vai a 7 altrimenti a 10	
7	Salva rip e rflags nella pila puntata da rsp	
8	Metti flip/flip I a 0	
9	Carica in rip l'indirizzo iniziale del programma di servizio	
10	Torna al passo 1 fino a che trovi una istruzione HLT	

Salvataggio dello Stato

Stato di un processo: contenuto dei registri del processore e delle locazioni di memoria usati dal codice del processo.

Le locazioni di memoria possono essere protette assegnando a ciascun processo una partizione distinta della memoria.

I registri interni del processore, invece, sono visibili a tutti i processi. Quindi necessità di memorizzare i contenuti dei registri che potrebbero essere modificati da altri processi (vedi routine di servizio).

Organizzazione a pila (stack di sistema - gestione LIFO) della memoria in cui andare memorizzare i contenuti dei registri (vedere gestione delle subroutine). Ricordarsi che il registro RSP è visto come STACK POINTER dello stack di sistema.

Possibilità di memorizzazione:

- via firmware tutti i registri
- via firmware solo RIP e SR, a software solo quelli che verranno effettivamente modificati.

Nello z64 si è optato per la seconda soluzione.

Interfaccia periferica

Per poter interagire con il processore la periferica deve presentare:

Un flip/flop che avverte il processore che vuole interagire con una interruzione (INT_REQ), questo flip/flop è settato dallo SCO della periferica e resettato dal processore quando riconosce che il dispositivo lo ha interrotto.

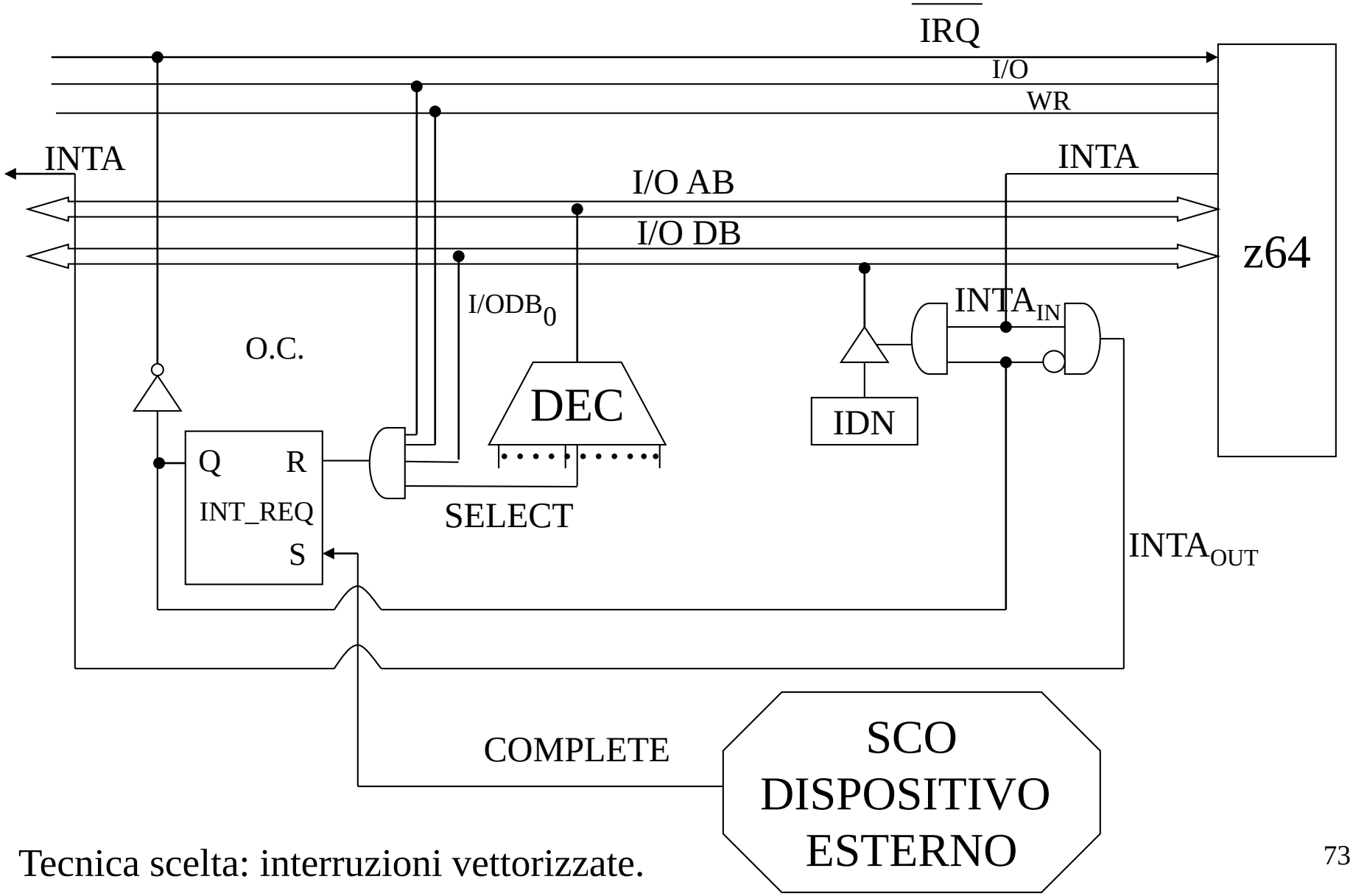
Un registro in cui memorizzare l'Interrupt Descriptor Number (IDN) che il processore utilizzerà per identificare l'inidirizzo iniziale del DRIVER da attivare a fronte della richiesta di interruzione. registri interni del processore, invece, sono visibili a tutti i processi. Quindi necessità di memorizzare i contenuti dei registri che potrebbero essere modificati da altri processi (vedi routine di servizio).

Un flip/flop utilizzato dal processore per avvertire la periferica che desidera interagire con essa, la quale una volta pronta per interagire interrompe il processore tramite un interrupt. Tale flip/flop è quello visto nell'acquisizione dati in busy waiting (STATUS), ma non verrà utilizzato dal processore in lettura per verificare che la periferica è pronta per l'interazione perché viene avvertito tramite interrupt.

Necessità di annullare la richiesta di interruzione

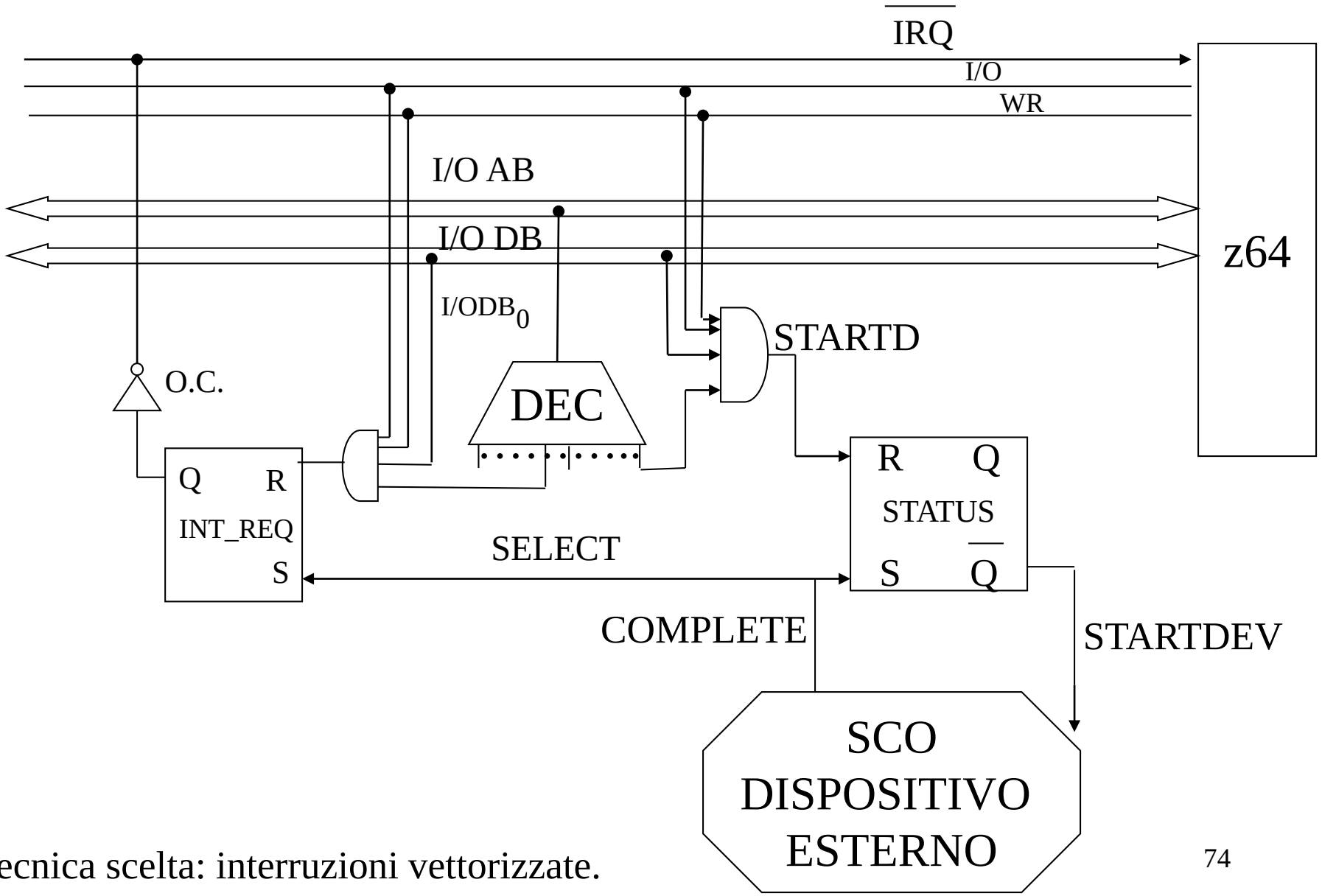
Una volta attivato il driver è necessario, all'interno del driver, annullare la richiesta dell'interruzione stessa. Poiché la richiesta di interruzione è memorizzata dalla periferica nel flip/flop INT_REQ il processore deve essere in grado di resettare tale flip/flop. Lo potrà fare con una istruzione di out. A tal fine sarà necessario aver scritto prima nel registro %dx l'indirizzo del flip/flop INT_REQ e nell'accumulatore il valore 1. All'atto dell'esecuzione dell'istruzione **out** verrà posto ad 1 il segnale di controllo I/O, e il segnale di controllo WR, mentre il bit meno significativo del Data Bus sarà pari ad 1 e sull'I/OAB ci sarà l'indirizzo di INT_REQ della periferica con cui il processore sta interagendo. Come si vede dalla Figura successiva il flip/flop di INT_REQ viene resettato dal segnale che viene generato dall'uscita della porta logica AND il cui ingresso è dato dal segnale di I/O, dal segnale WR, da DB0 e dal "select" generato a fronte della presenza dell'indirizzo della periferica presente sull'I/OAB. In questo modo si annulla la presenza dell'interruzione ormai riconosciuta dal processore.

Identificazione programma di servizio



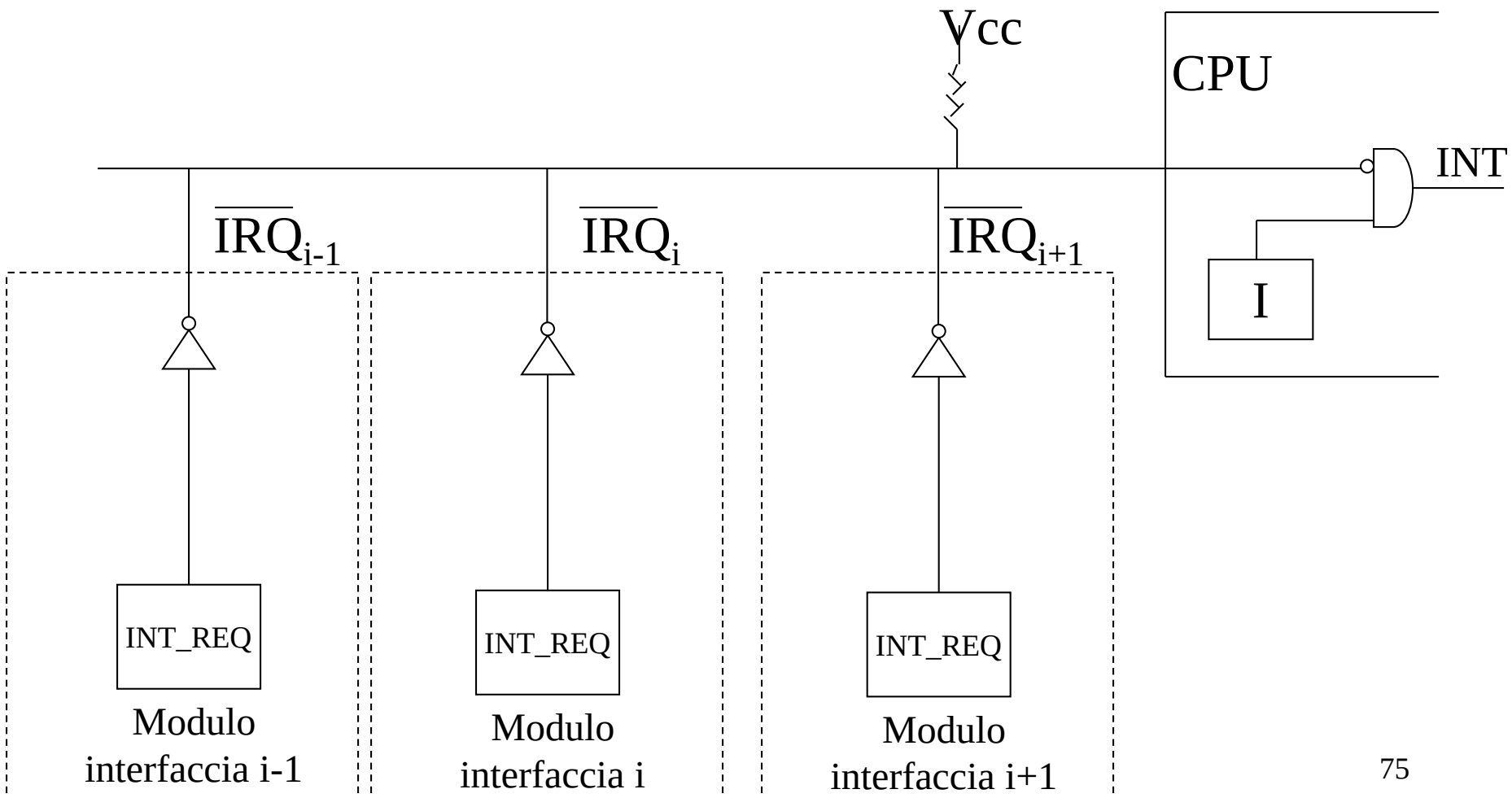
Tecnica scelta: interruzioni vettorizzate.

Inserimento del flip/flop STATUS



Tecnica scelta: interruzioni vettorizzate.

Connessione, in wired OR, di più interfacce alla linea IRQ



- Completamento salvataggio dello stato
- esecuzione del programma di servizio
- ripristino stato

- Salvataggio del contenuto dei registri (visibili dall'utente) che la routine di servizio modificherà tramite esecuzione di istruzioni tipo PUSH.

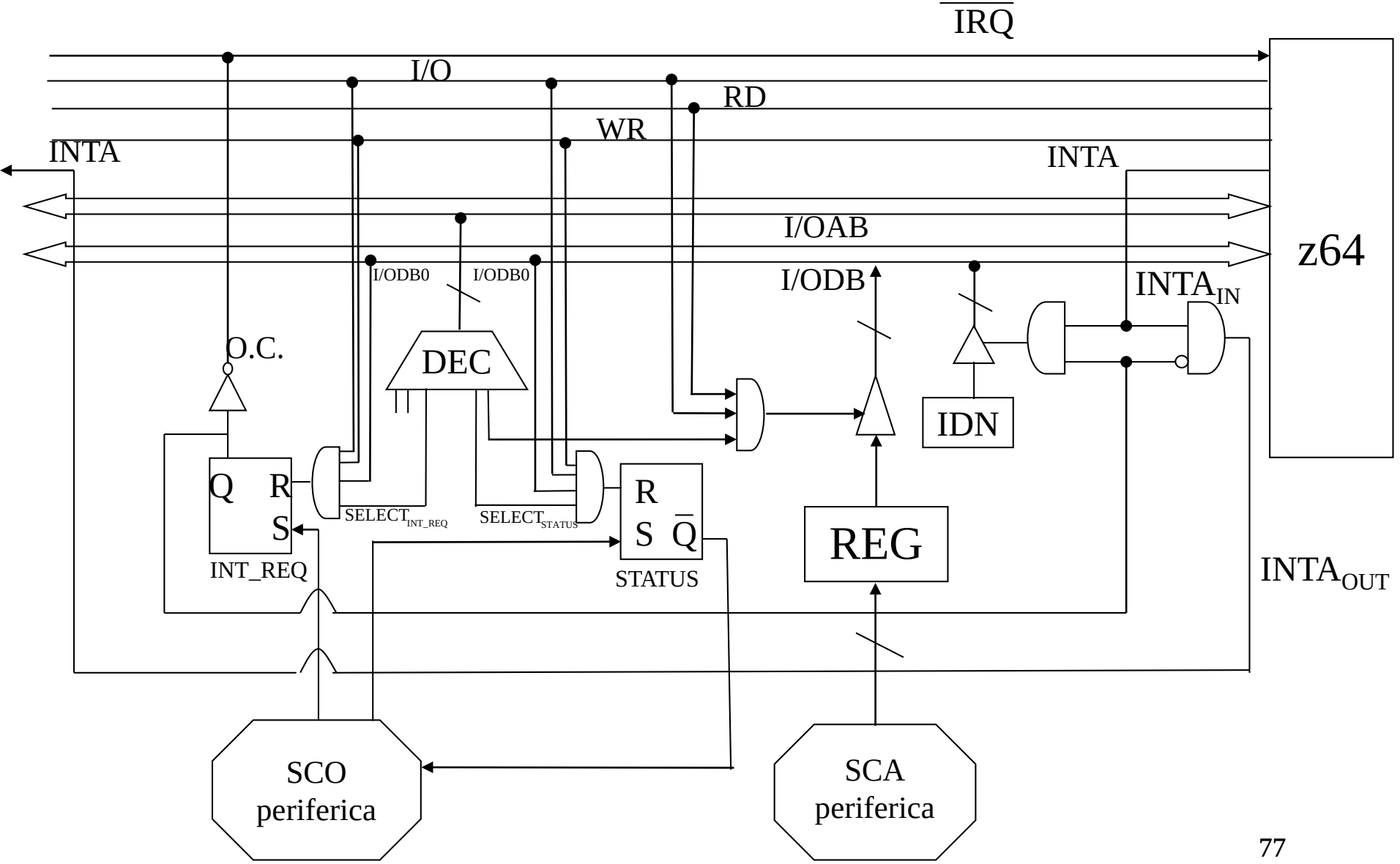
- Esecuzione programma.

- Ripristino del contenuto dei registri salvati nello stack, tramite POP (tante POP per quante PUSH effettuate precedentemente).

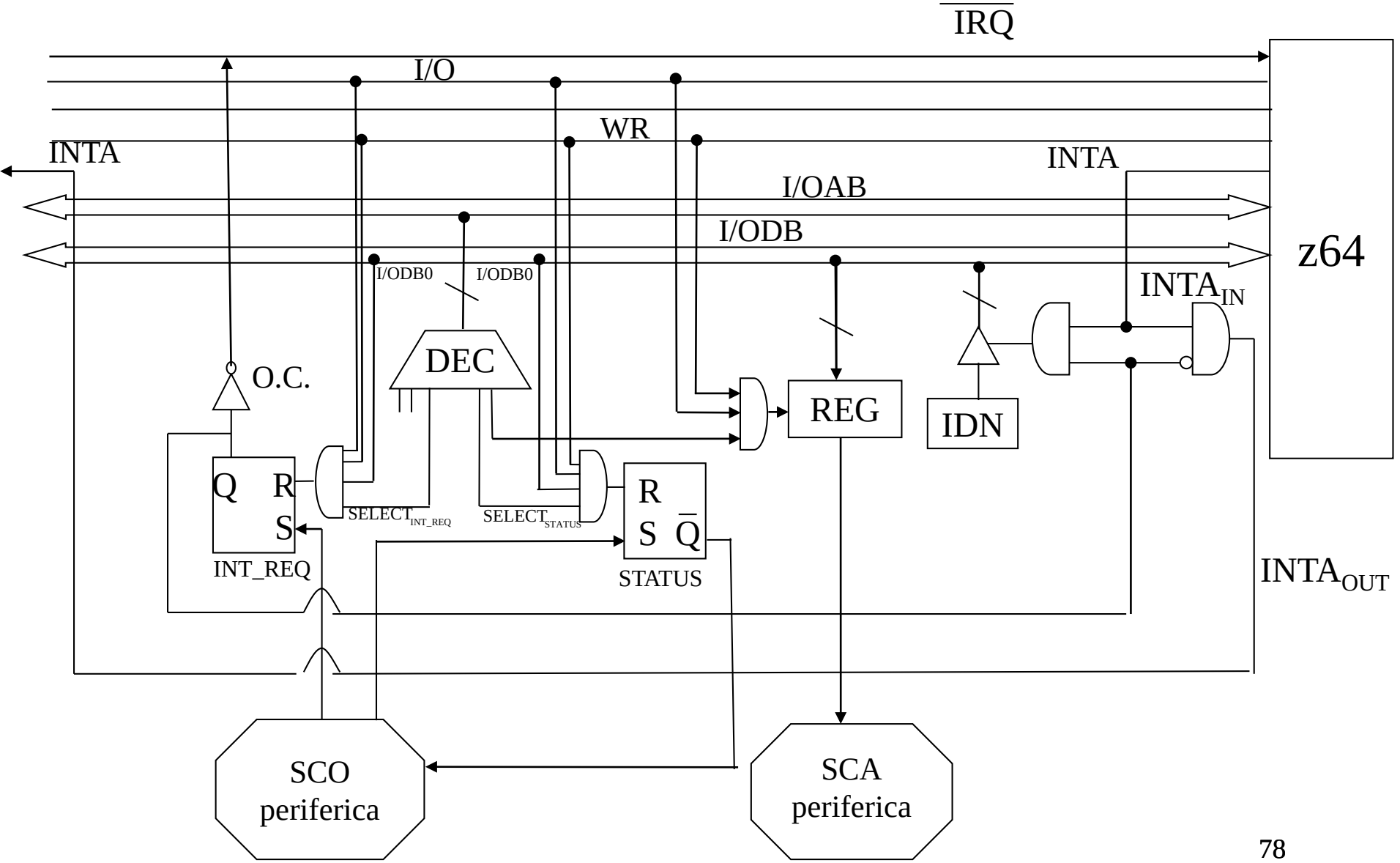
- Esecuzione della IRET (**I**nterrupt **RET**urn), che ripristina in FLAGS e in RIP i valori memorizzati nello stack di sistema (equivalente a due istruzioni POP).

IRET quindi deve essere l'ultima istruzione della routine di servizio.

Interfaccia per *lettura* dati da periferica con interrupt



Interfaccia per *scrittura* dati su periferica con interrupt



Driver: acquisizione di 100 dati tramite interruzione (1/2)

```
.org 0x800                    # Memorizza il programma dopo l'IDT
  .equ dati, 0xAAAA           # Indirizzo di memoria da dove iniziare a memorizzare i dati
  .equ AD, 0xAA               # Indirizzo registro da dove acquisire dato
  .equ STATUS_AD, 0xAB
  .equ INT_REQ_AD, 0xAC
.text                         # Identifica l'inizio del programma
   xorl %rcx, %rcx            # Resetta il contatore dei dati acquisiti
   movq $dati, %rdi           # Imposta il 'registro destinazione' con la base del vettore
   movw $STATUS_AD, %dx
   movb $1, %al
   outb %al, %dx              # Avvia la periferica
   sti                        # Abilita lo z64 per ricevere interruzioni
.loop:
   hlt                        # pone lo z64 in attesa di un interrupt
   jmp .loop                  # notare che rimane in attesa di interrupt anche dopo aver
                              # acquisito i 100 dati
```

Driver: acquisizione di 100 dati tramite interruzione (2/2)

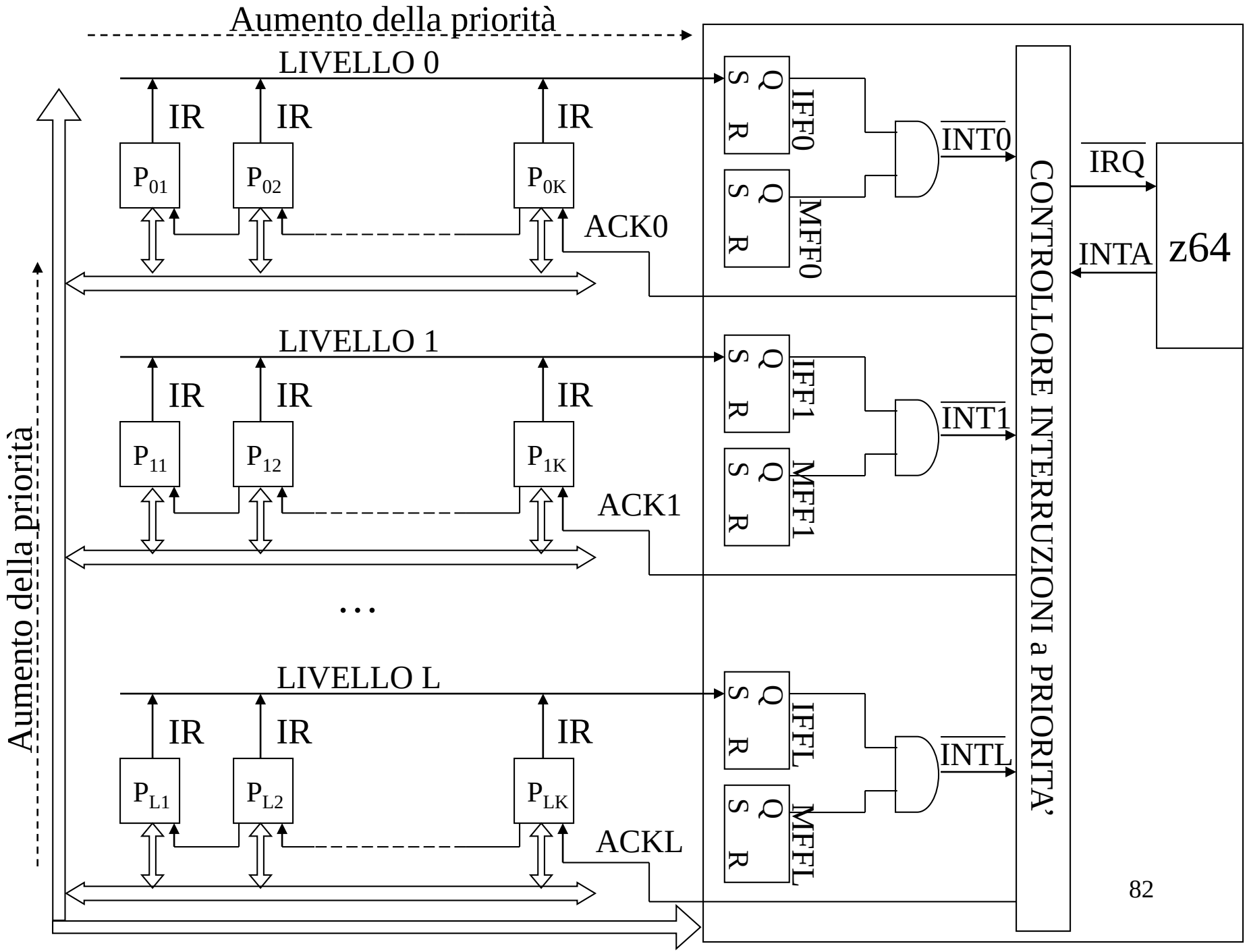
```
.driver 1                       # ‘1’ è l’IDN della periferica
   movw $AD, %dx                # AD è l’indirizzo del registro di porta di I/O
   inw %dx, %ax                 # acquisisce una word dalla periferica
   movw %ax, (%rdi, %rcx, 2)  # copia il dato acquisito nel vettore in memoria
   movw $INT_REQ_AD, %dx # INT_REQ_AD è l’indirizzo del F/F di interruzione
   movb $1, %al                 # Scrivere ‘1’ su INT_REQ_AD cancella la causa di interruzione…
   outb %al, %dx                # elimina la richiesta di interruzione
   addl $1, %rcx                # incrementa il contatore
   cmpq $100, %rcx              # verifica se sono state fatte 100 acquisizioni
   jz .exit                     # in caso affermativo, esce direttamente dal driver
   movb $1, %al                 # altrimenti riavvia periferica
   movw $STATUS_AD, %dx # STATUS_AD è l’indirizzo del F/F di stato della periferica
   outb %al, %dx                # riavvia la periferica

  .exit:
   iret                         # ritorno da interruzione (all’istruzione dopo hlt)
.end
```

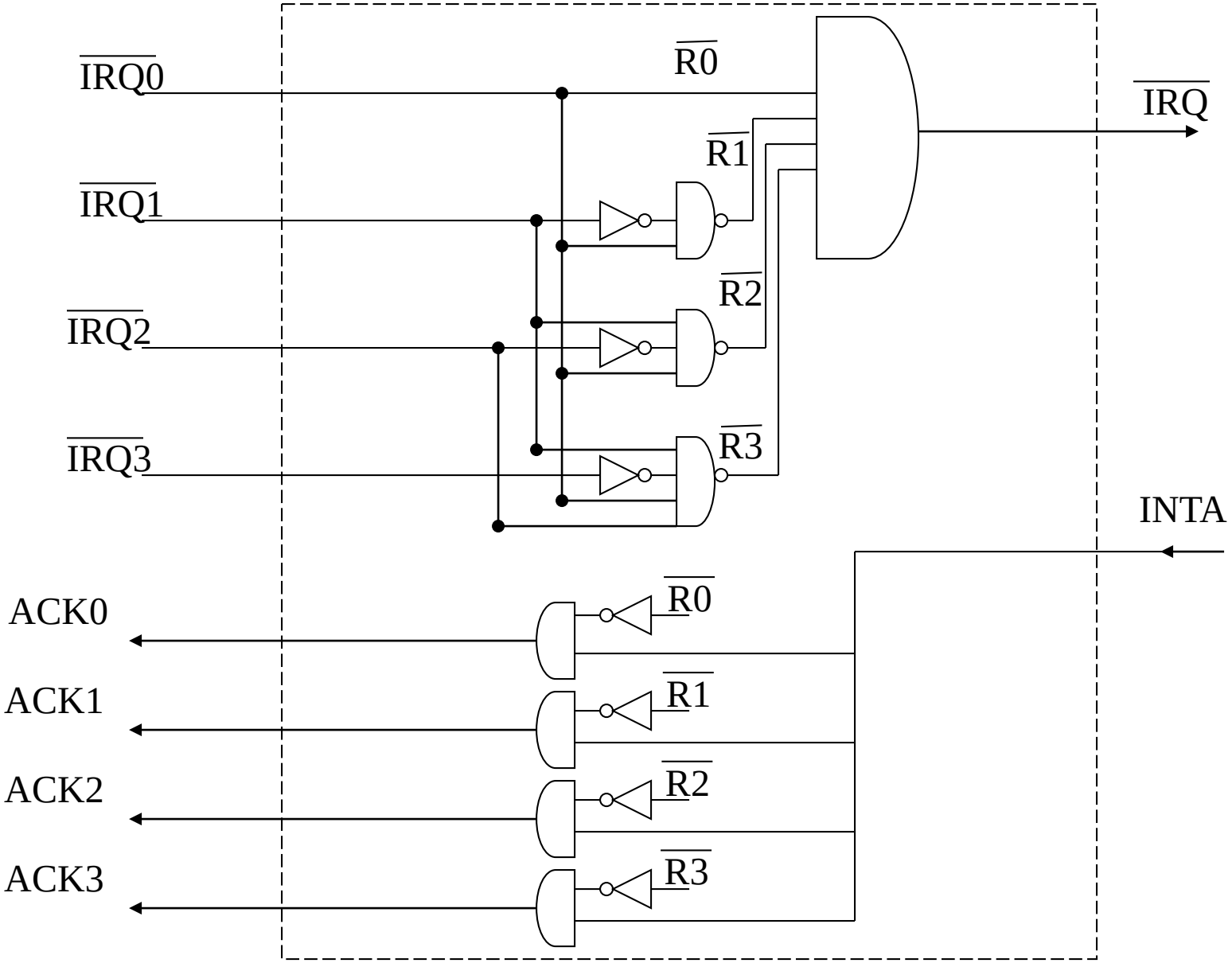
Concetto di azione atomica

Flip-Flop I: messo a 0 dal microprogramma relativo al ciclo riconoscimento interrupt

messo a 1 dal microprogramma relativo a IRET


Aumento della priorità
LIVELLO 0
IR
IR
IR
P01
P02
P0K
ACK0
Q
S
R
IFF0
MFF0
INT0
CONTROLLORE INTERRUZIONI a PRIORITA'
IRQ
INTA
z64
LIVELLO 1
IR
IR
IR
P11
P12
P1K
ACK1
IFF1
MFF1
INT1
...
LIVELLO L
IR
IR
IR
PL1
PL2
PLK
ACKL
IFFL
MFFL
INTL
Aumento della priorità

Controllore interruzione a priorità



Costo di esecuzione del driver per il trasferimento dati tramite interrupt di un file (100 dati) da una periferica ad un processore

Dall’esempio precedente calcolare

numero di istruzioni e quindi numero di cicli di clock complessivi

per eseguire un trasferimento

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

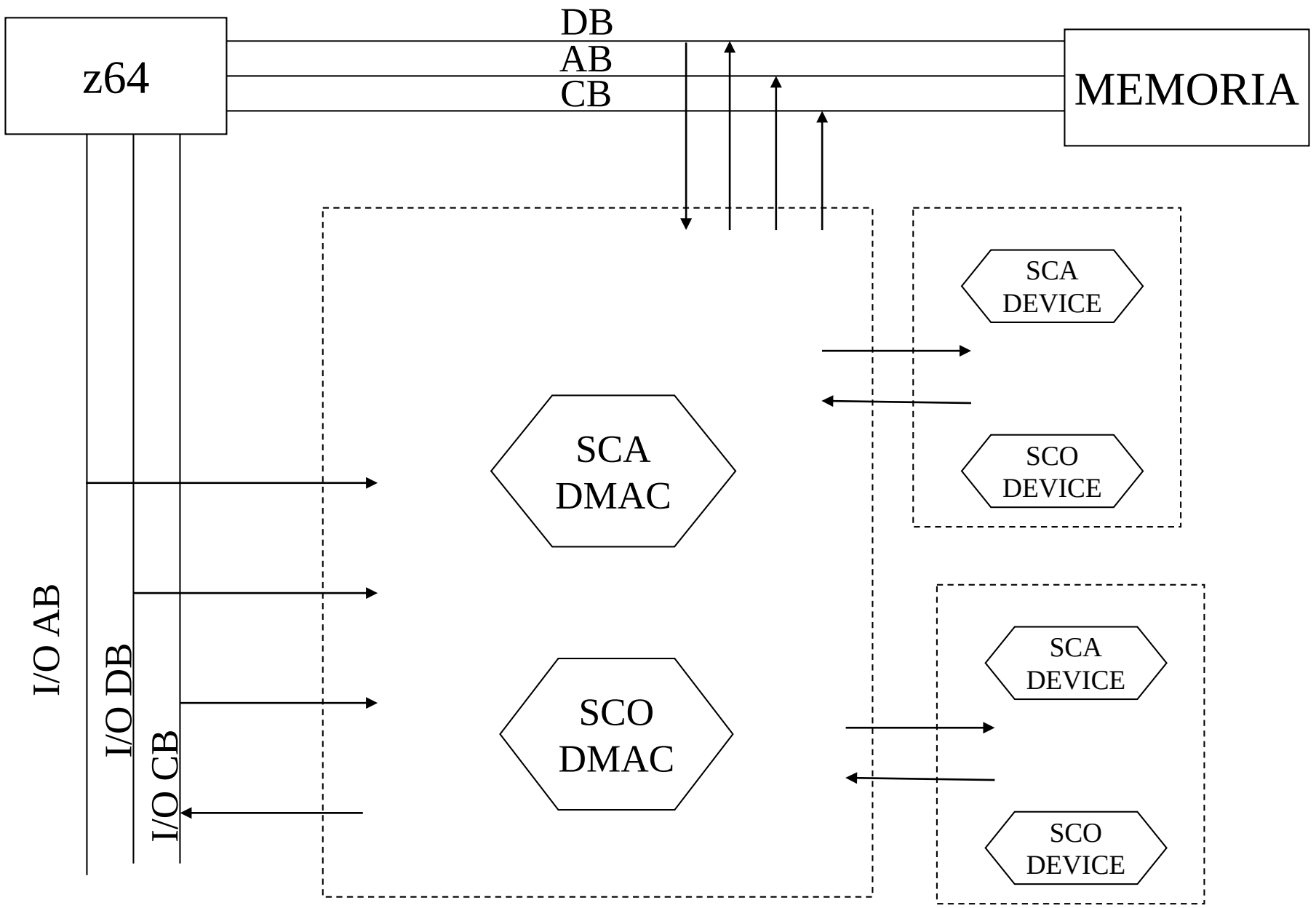
Una volta che il DMAC è stato programmato il processore lo deve attivare (p.e. tramite una START)

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**.

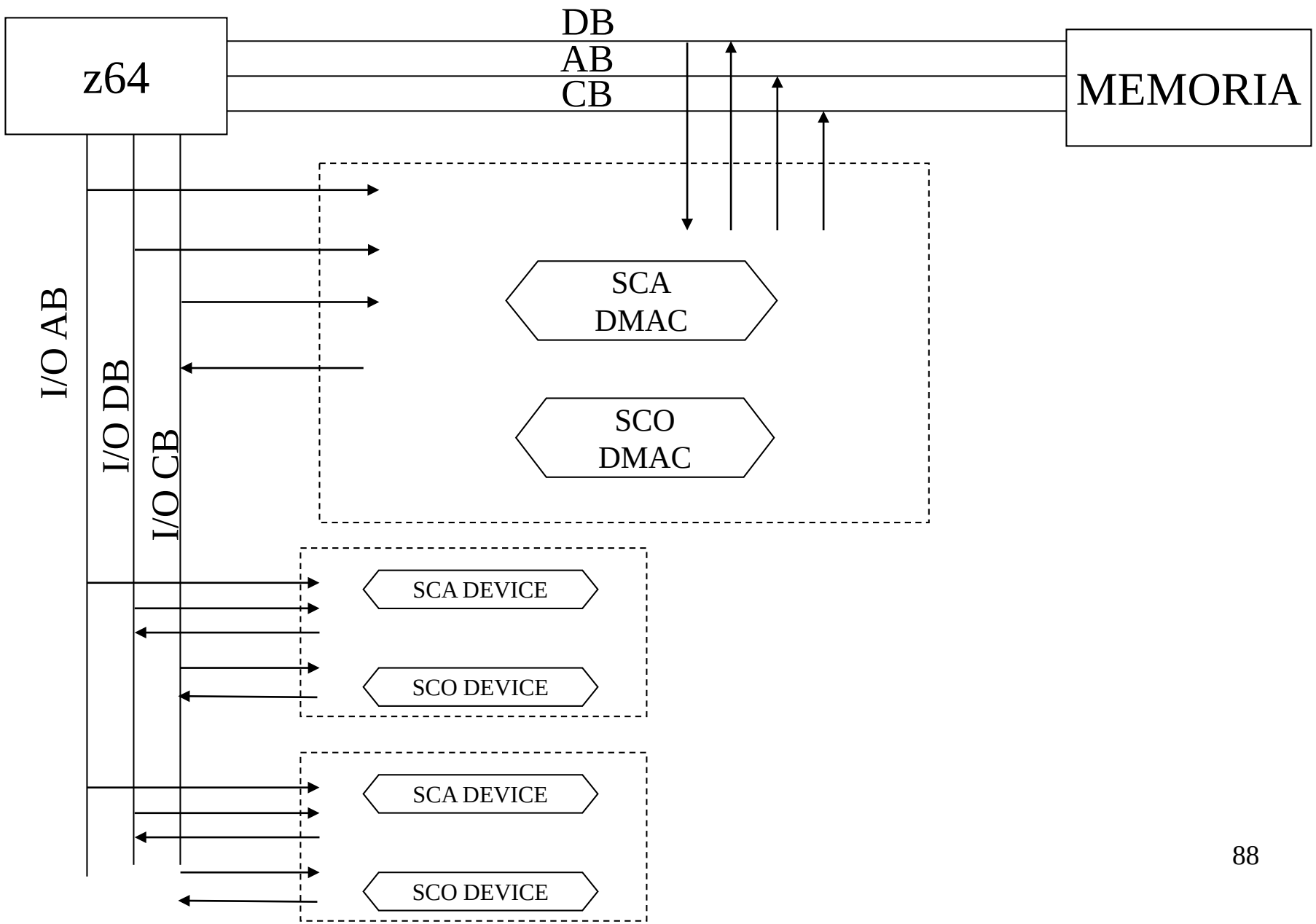
Una volta che il DMAC ha effettuato il trasferimento dei dati così come richiestogli dalla CPU la deve avvertire (p.e. tramite INTERRUPT).

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

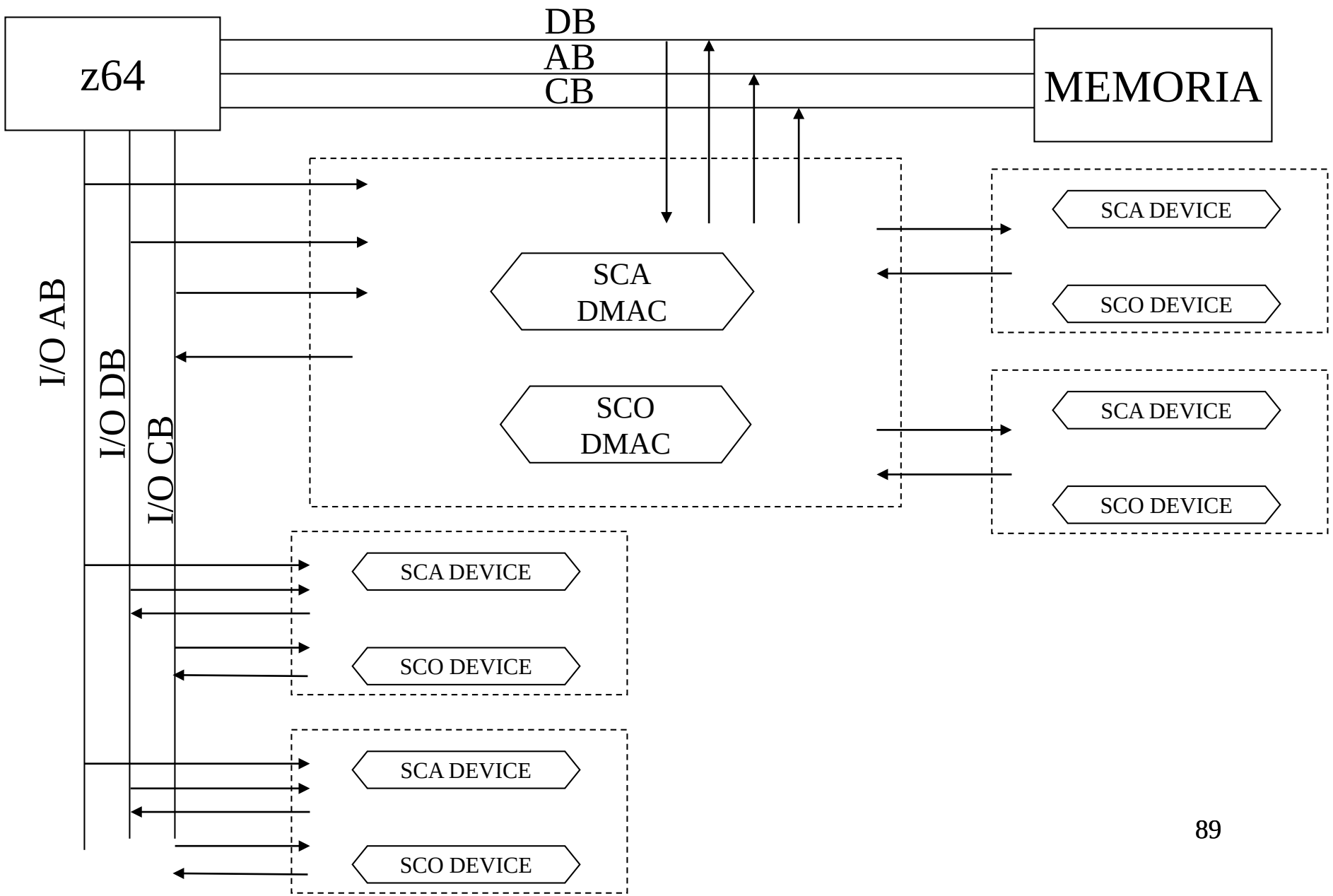
Struttura semplificata di un DMA controller
(periferiche non visibili dal processore)



Struttura semplificata di un DMA controller (periferiche visibili dal processore)



Struttura semplificata di un DMA controller (con periferiche visibili o meno dal processore)



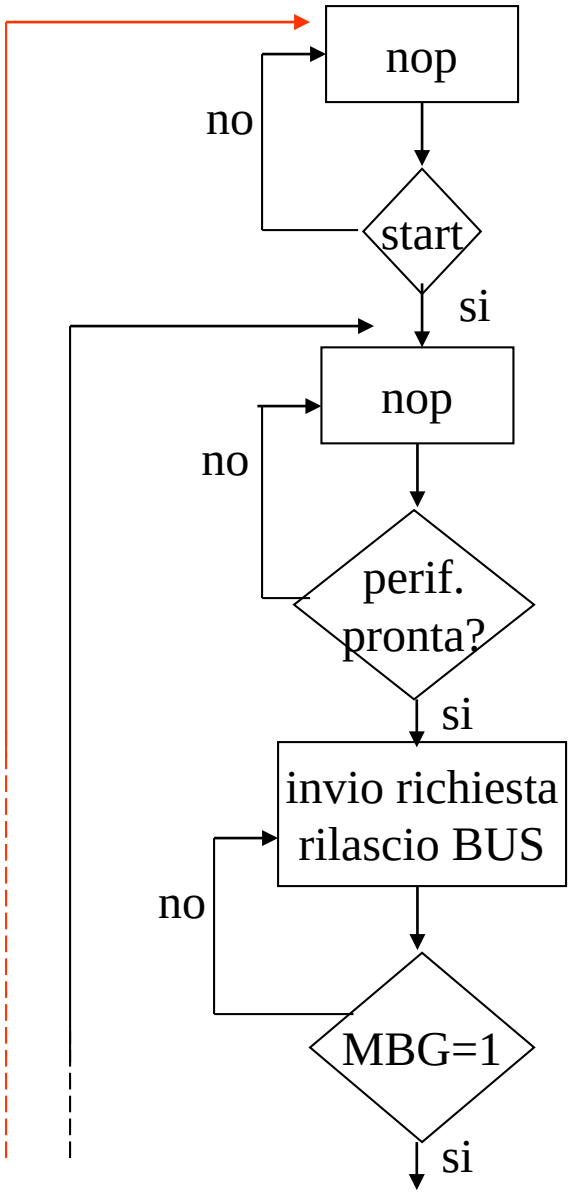
Protocollo di interazione DMAC-CPU

Trasferimento a Bus - stealing

DMAC

CPU

(1/2)



I/O

I/O

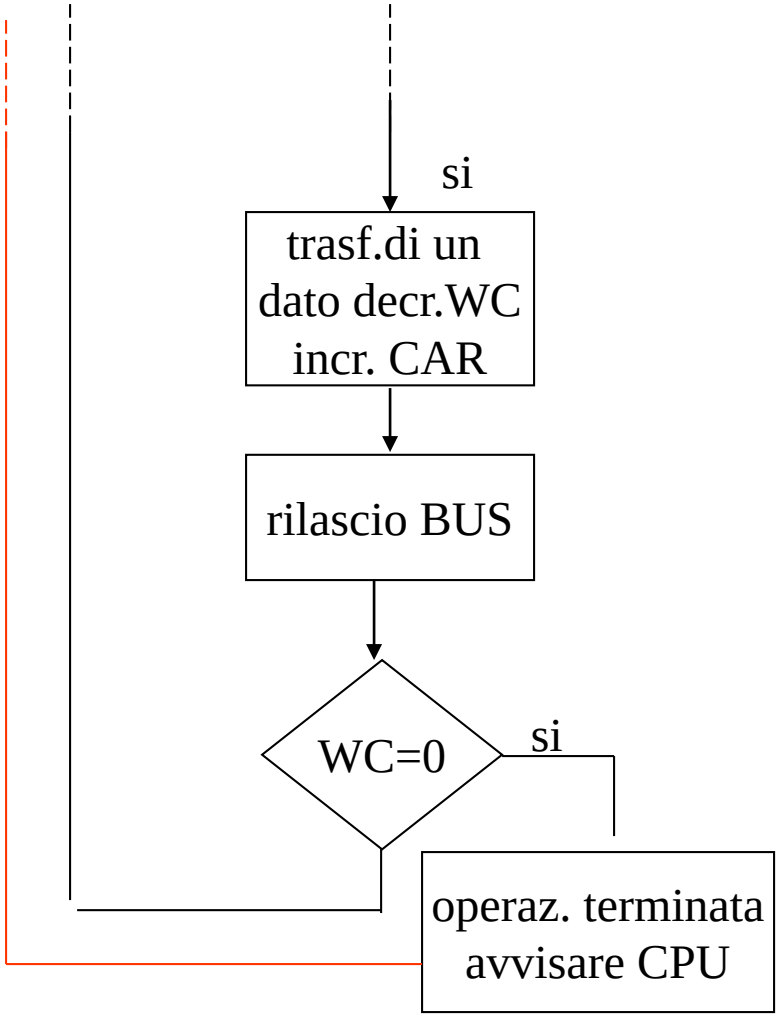
START

MBR

MBG

La CPU
Inizializza
Il DMAC

La CPU termina il ciclo macchina ed entra in uno stato di sospensione “rilascio dei bus” uscite in alta impedenza



MBR

IRQ

La CPU riprende il normale funzionamento *Riacquisto del controllo del BUS*

La CPU esegue la routine di servizio

DMAC

CPU

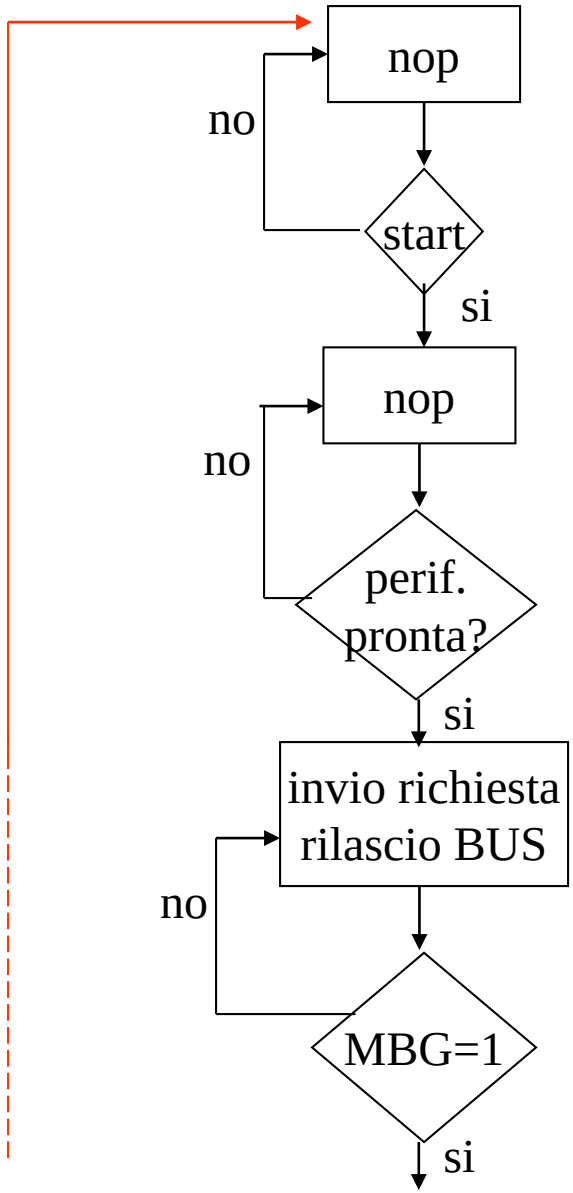
Protocollo di interazione DMAC-CPU

Trasferimento a BURST

DMAC

CPU

(1/2)



I/O

I/O

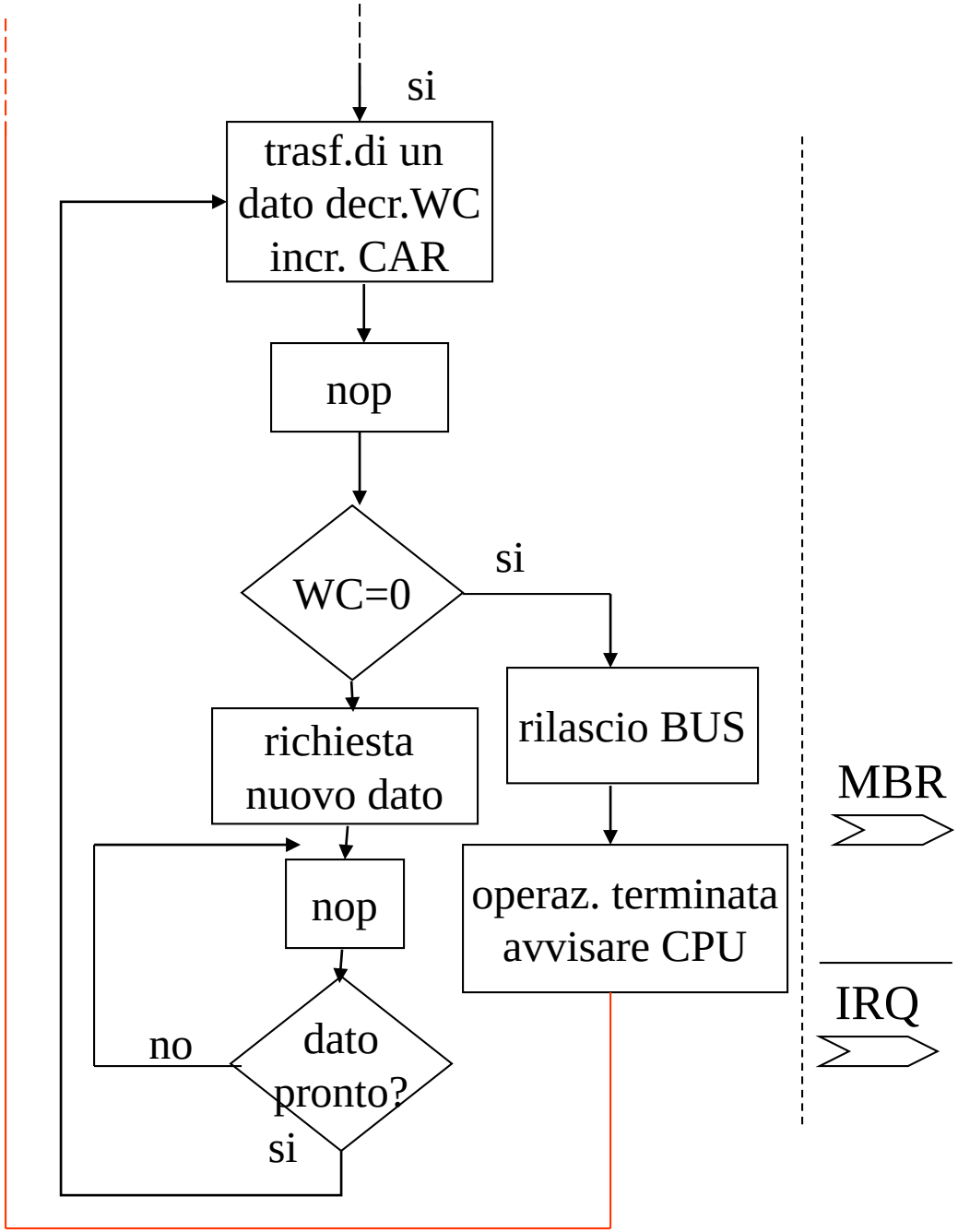
START

MBR

MBG

La CPU
Inizializza
Il DMAC

La CPU termina il ciclo macchina ed entra in uno stato di sospensione “rilascio dei bus” uscite in alta impedenza



MBR

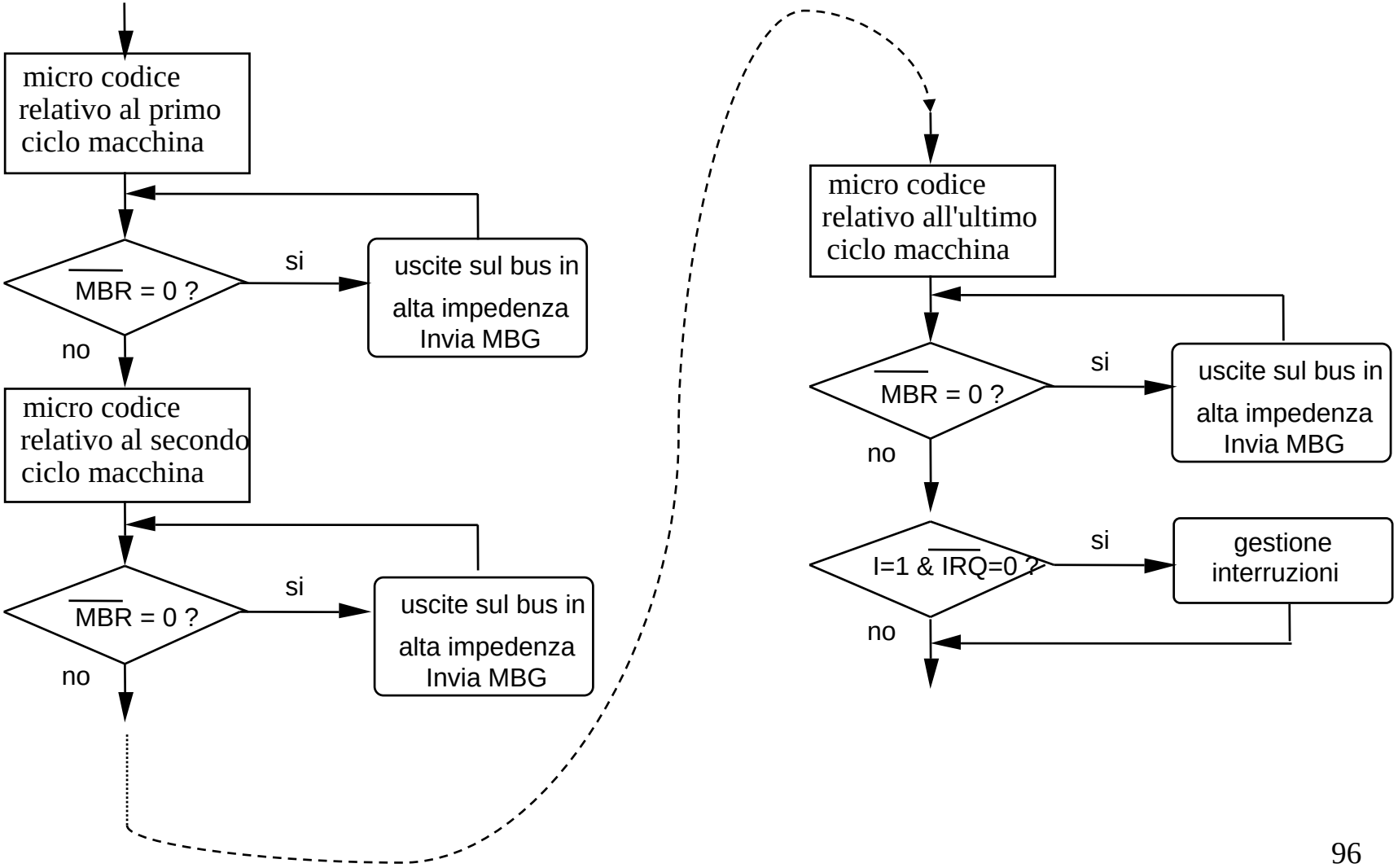
IRQ

La CPU riprende il normale funzionamento *Riacquisto del controllo del BUS*

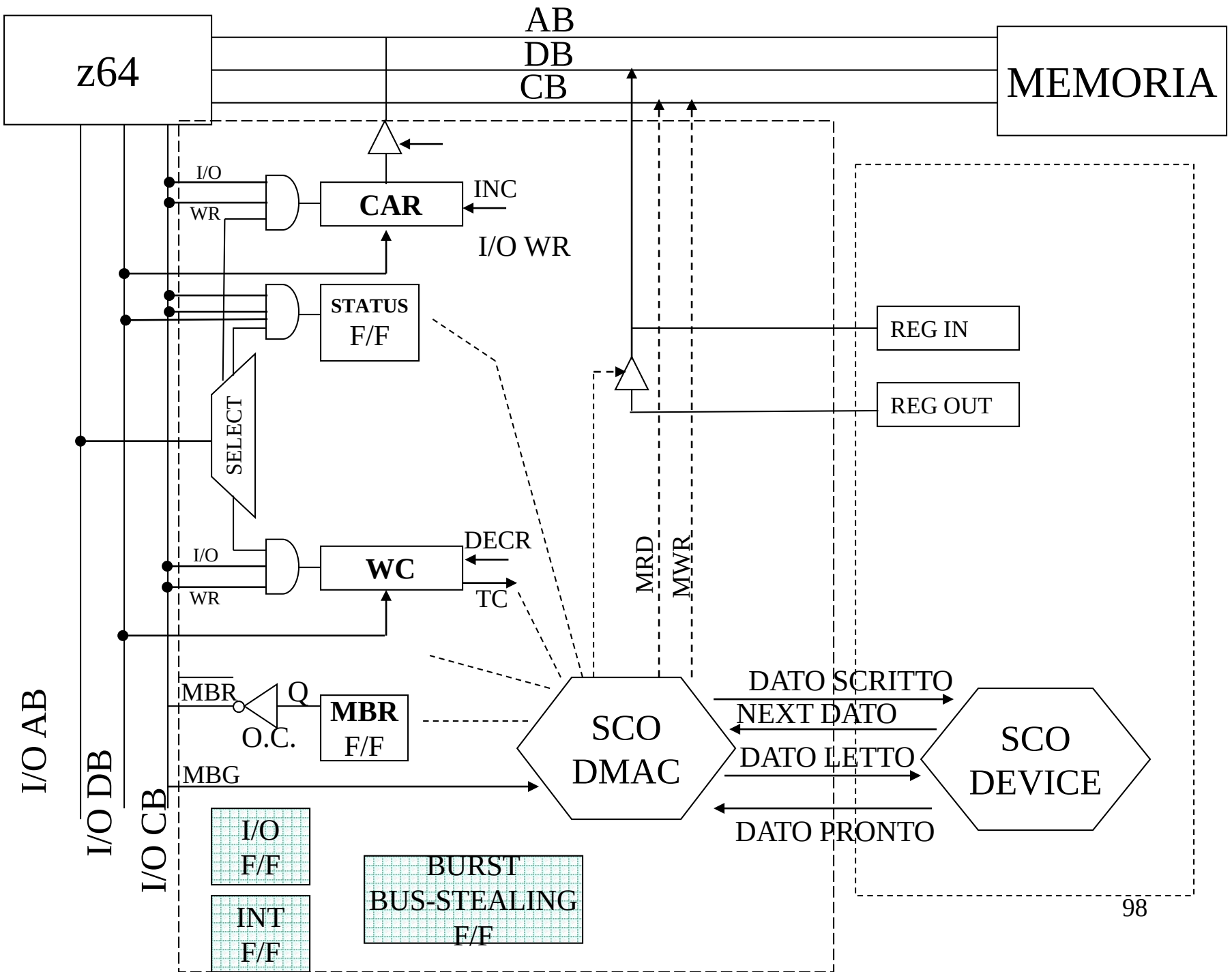
La CPU esegue la routine di servizio

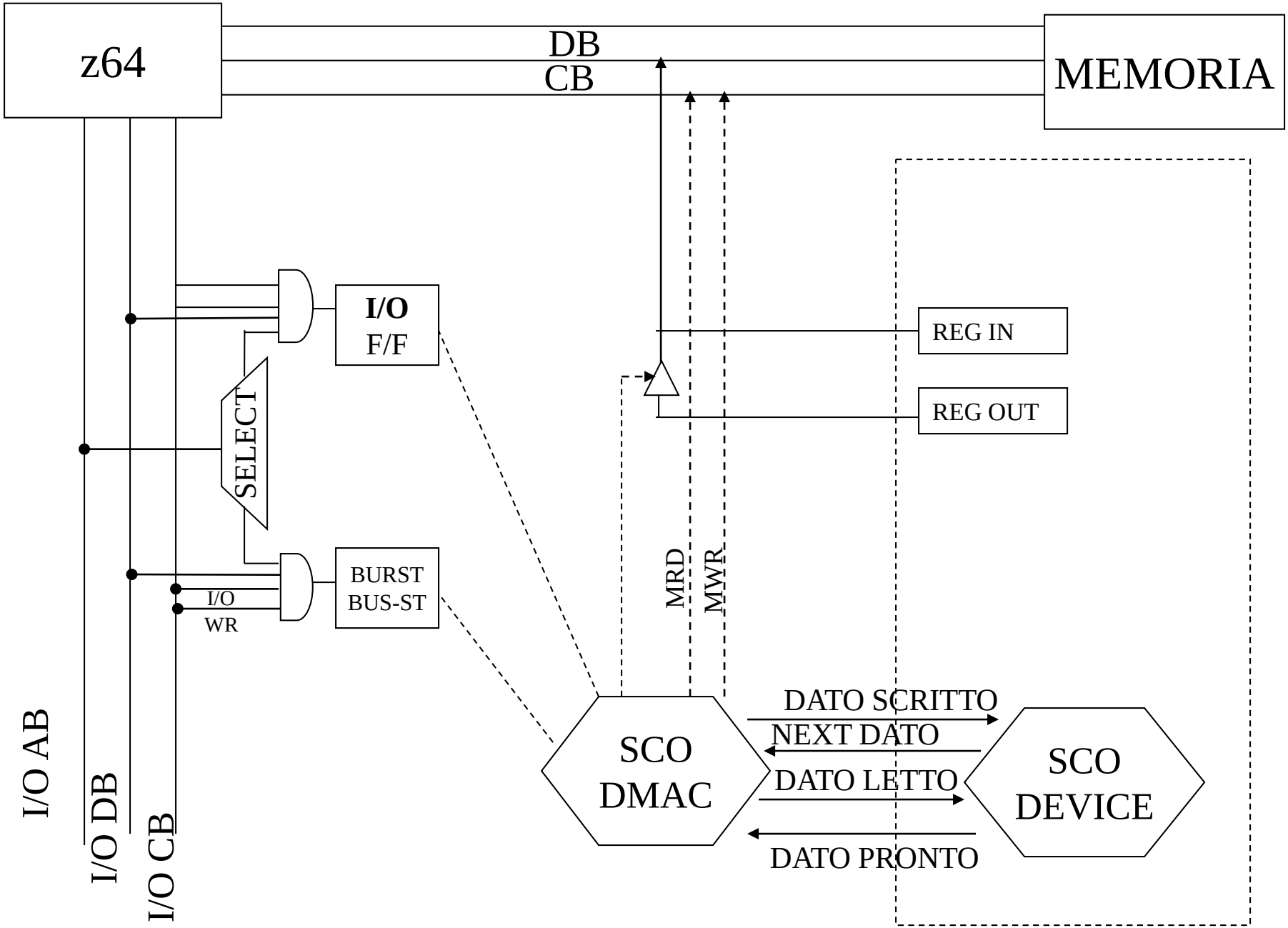
CPU

Modifica SCO z64
per poter gestire le due modalità di interazione

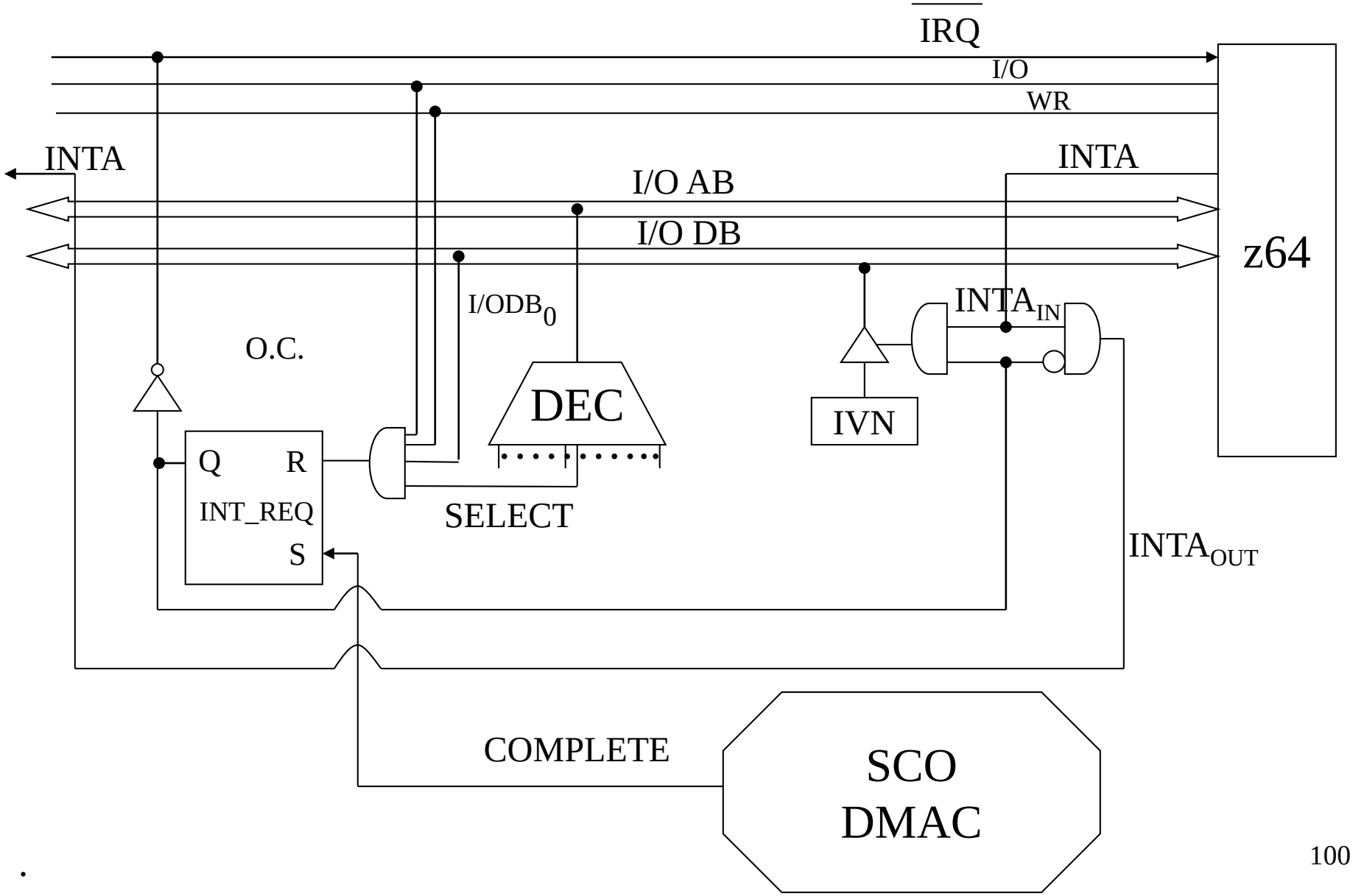


Architettura di un DMAC (adatto x z64) trasferimento dati tra una periferica e la memoria


z64
AB
DB
CB
MEMORIA
I/O
WR
CAR
INC
I/O WR
STATUS
F/F
REG IN
REG OUT
SELECT
I/O
WR
WC
DECR
TC
MRD
MWR
MBR
Q
O.C.
MBR
F/F
MBG
I/O AB
I/O DB
I/O CB
SCO
DMAC
DATO SCRITTO
NEXT DATO
DATO LETTO
DATO PRONTO
SCO
DEVICE
I/O
F/F
INT
F/F
BURST
BUS-STEALING
F/F

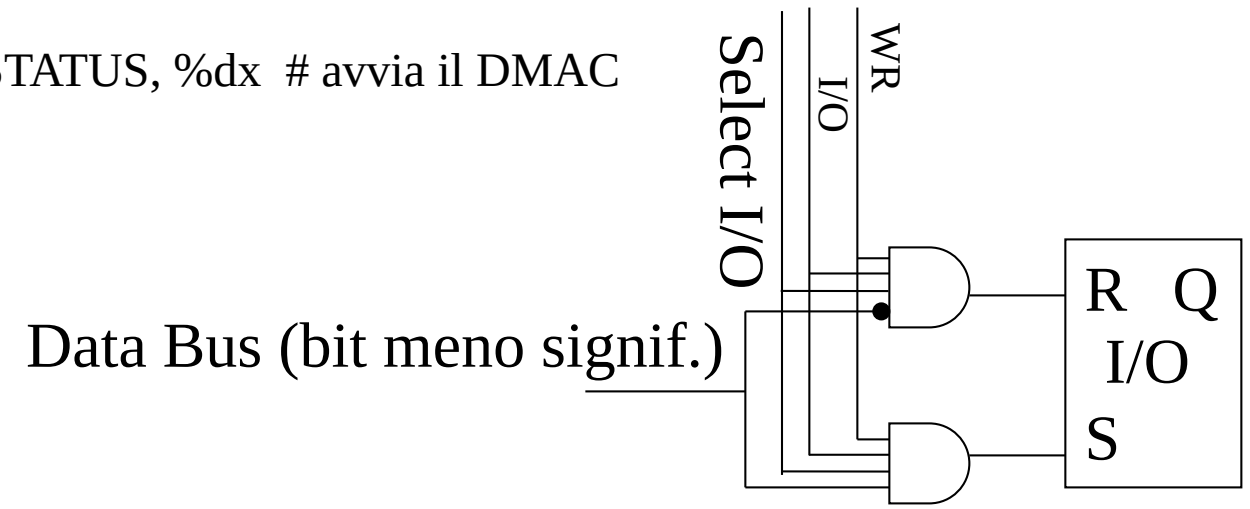

z64
DB
CB
MEMORIA
I/O
F/F
SELECT
BURST
BUS-ST
I/O
WR
REG IN
REG OUT
MRD
MWR
I/O AB
I/O DB
I/O CB
SCO
DMAC
DATO SCRITTO
NEXT DATO
DATO LETTO
DATO PRONTO
SCO
DEVICE

Identificazione programma di servizio



Esempio inizializzazione DMAC

```
movw $WC, %dx              # inizializza WC a 100
movl $100, %eax
outl %eax, %dx
movw $CAR, %dx             # inizializza CAR a 0x800aaa
movl $0x800aaa, %eax
outl %eax, %dx
movw $DMACIO, %dx          # Inizializza il DMAC per la scrittura
movl $1, %eax
outl %eax, %dx
movw $DMACB-ST, %dx        # Inizializza il DMAC per lavorare in burst
movl $0, %eax
outl %eax, %dx
movw $DMAC_STATUS, %dx  # avvia il DMAC
movl $1, %eax
outl %eax, %dx
```



Implementazione dell’ins e dell’outs

programmando il dmac controller