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*
  < 10% ogni anno (IO per sec)
- Legge di Amdahl:

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

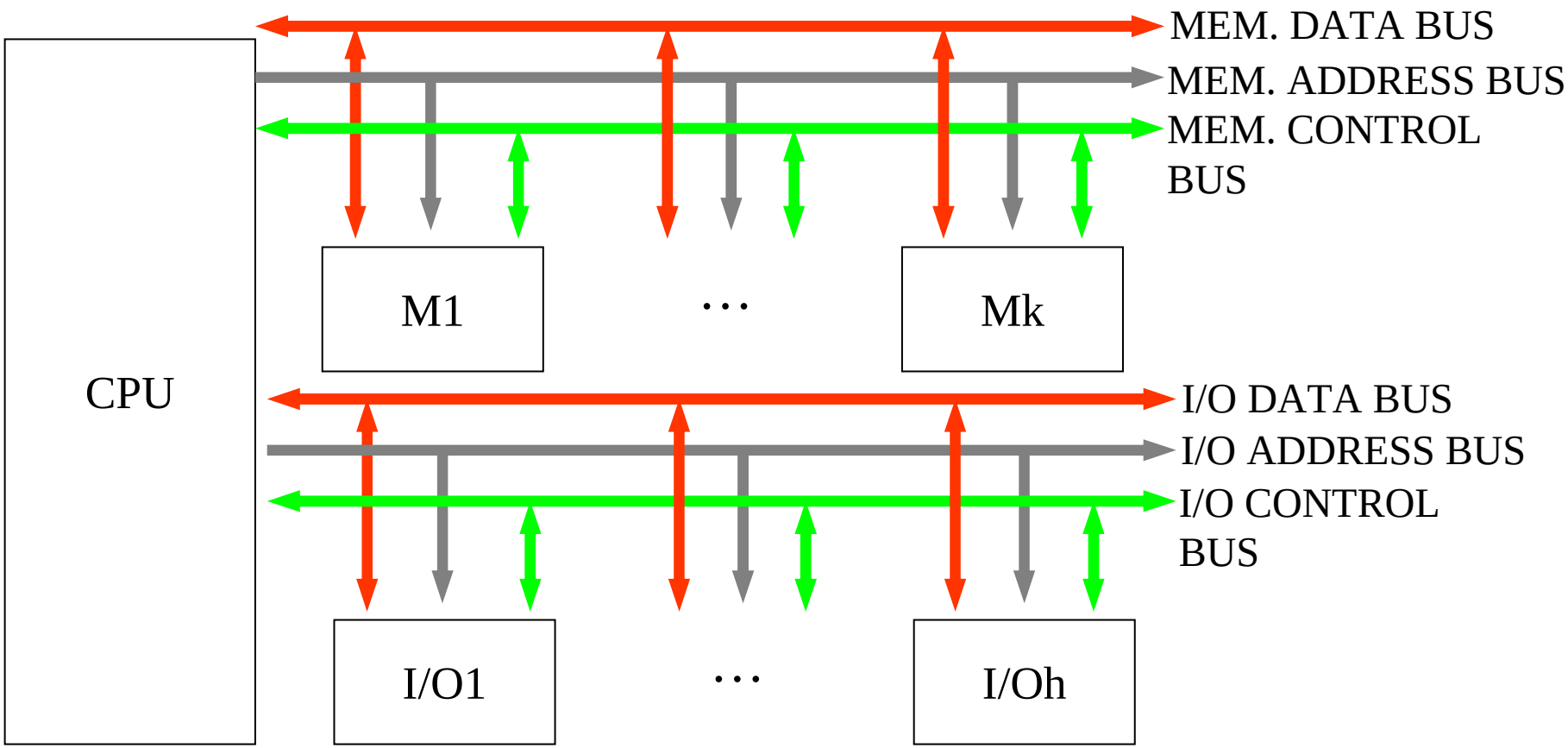
- S è lo speed-up effettivo
- f è la frazione di lavoro non dipendente dall'I/O
- K è lo speed-up della modalità "veloce"

- Lo speed-up di un sistema è limitato dal componente più lento:
  10% I/O & 10x CPU (f=0,9;k=10) => ≃5x Performance (- 50%)
  10% I/O & 100x CPU (f=0,9;k=100)=> ≃10x Performance (- 90%)

- Il sottosistema di I/O rappresenta un collo di bottiglia:
  < frazione di tempo in cui la CPU è effettivamente attiva
  < efficienza delle CPUs più veloci

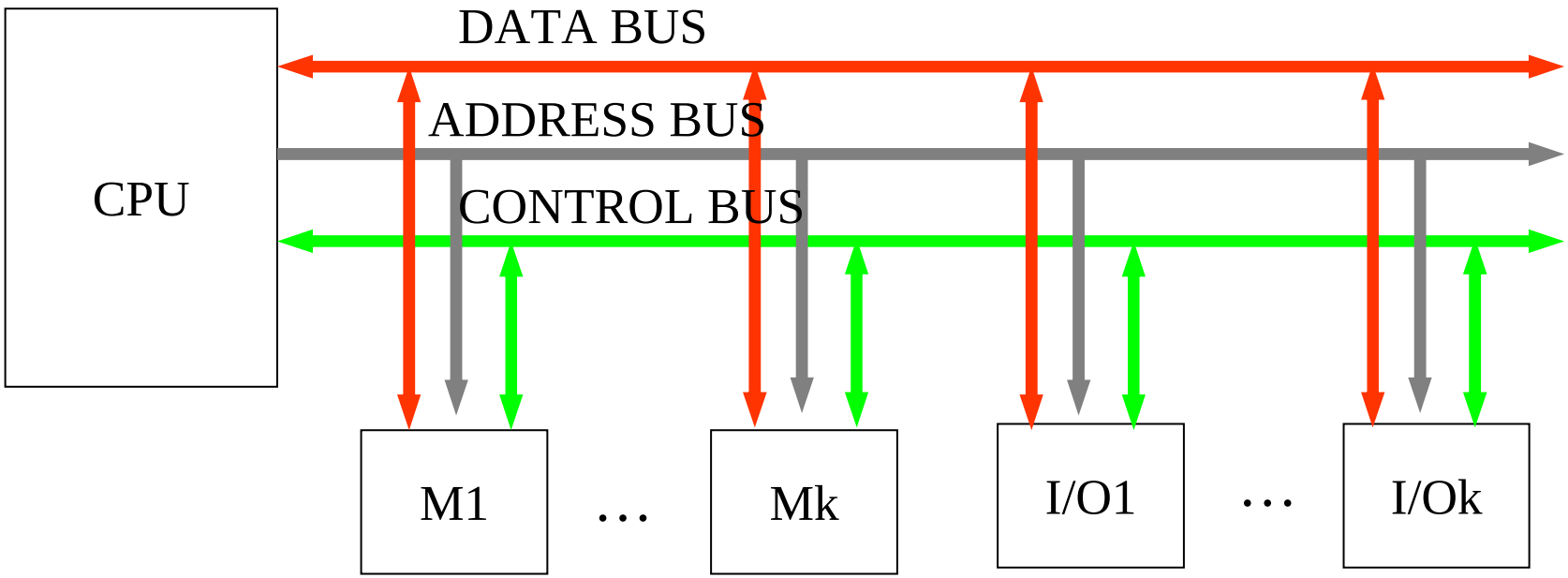
Possibili connessione tra CPU e dispositivi di Ingresso/Uscita

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



Possibili connessione tra CPU e dispositivi di Ingresso/Uscita

Architettura ad un solo bus



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

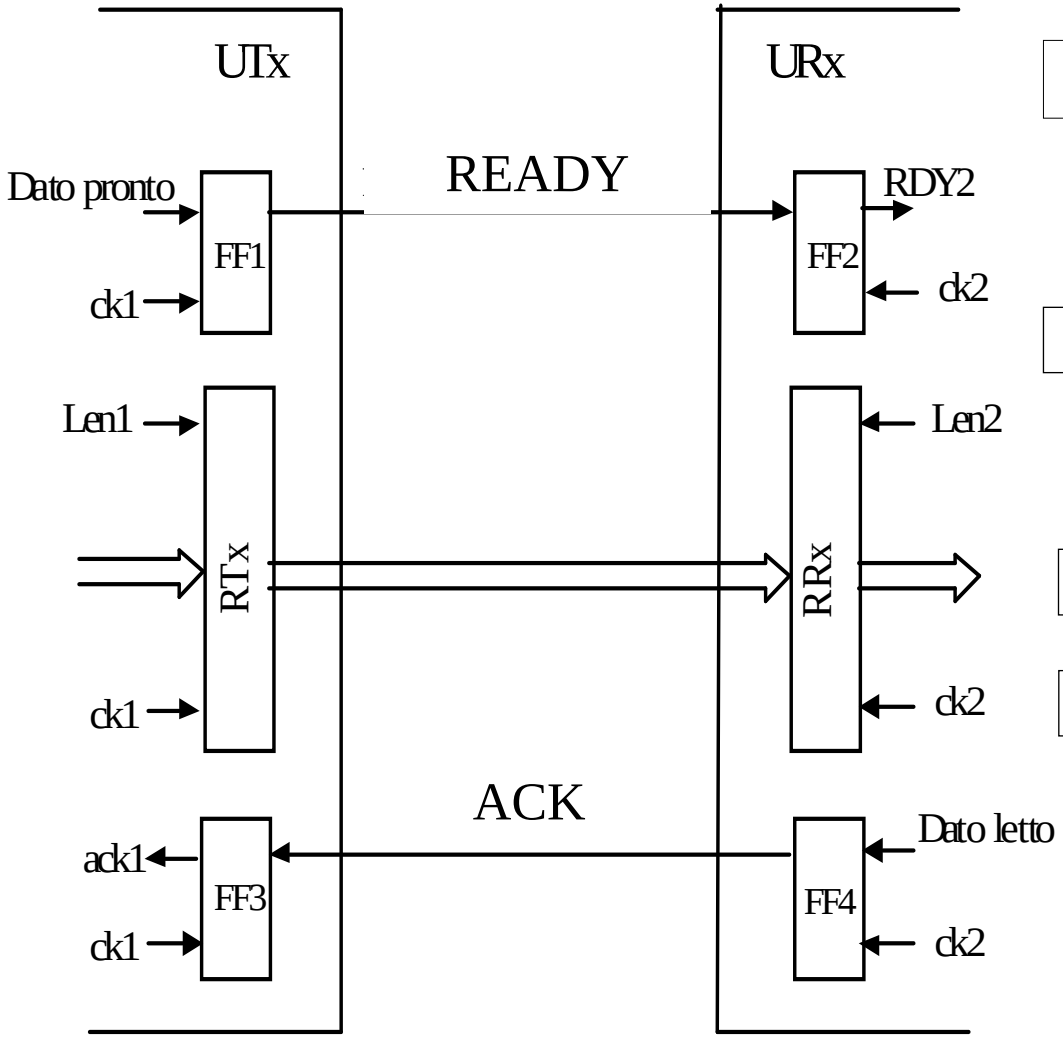
gestite da processori dedicati
- gestione delle comunicazioni tramite un servizio di segreteria
  (persona dedicata)

Tipi di interazione tra CPU e dispositivi esterni che studieremo

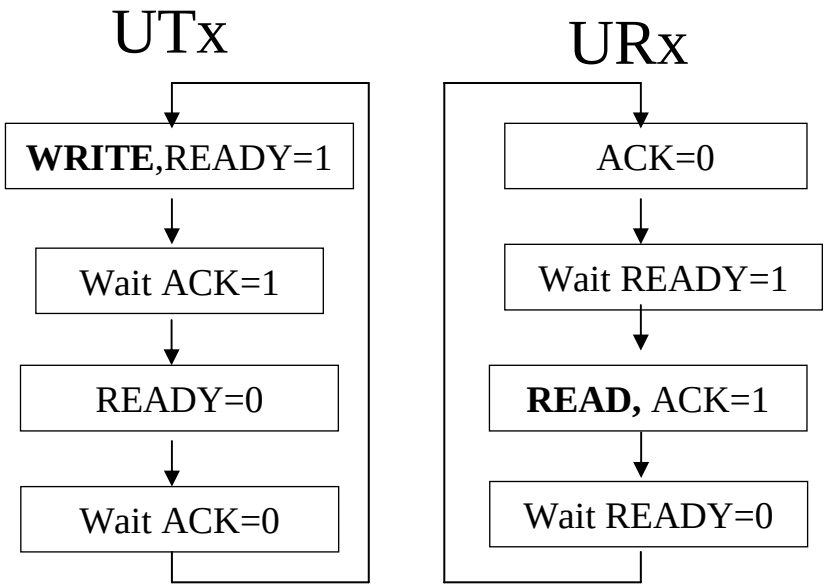
- Previste dai programmi che vengono eseguiti nella CPU (I/O programmato):
  - modalità busy waiting, implementata:
    i. a firmware
    ii. a software
  - modalità polling

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

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

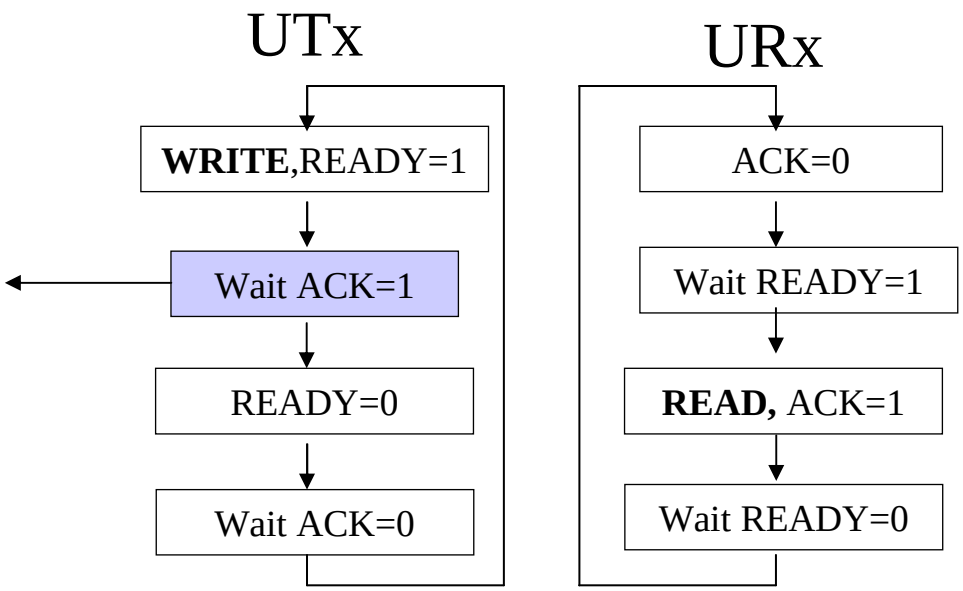


Initial values: RDY=0, ACK=0



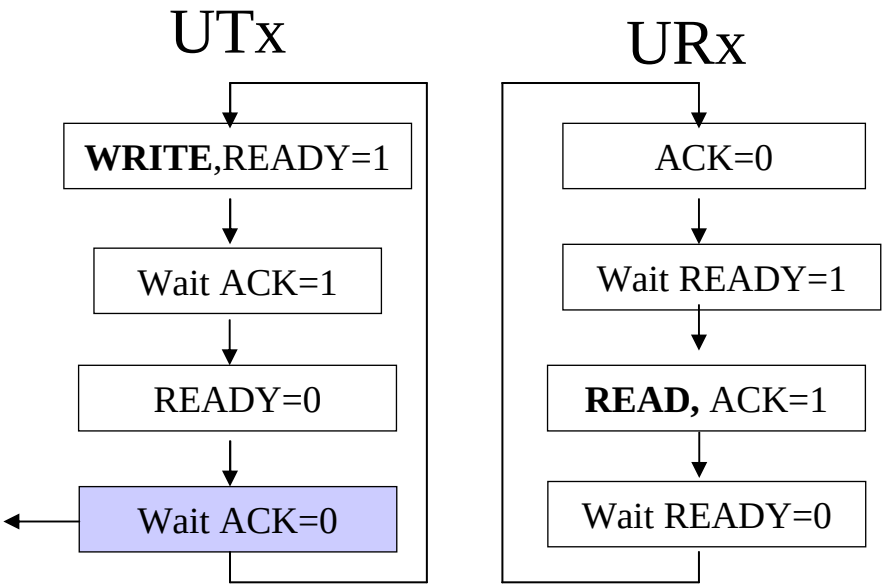
Che succede se si rimuovono le wait?

Sovrascrivo prima che la lettura sia avvenuta…

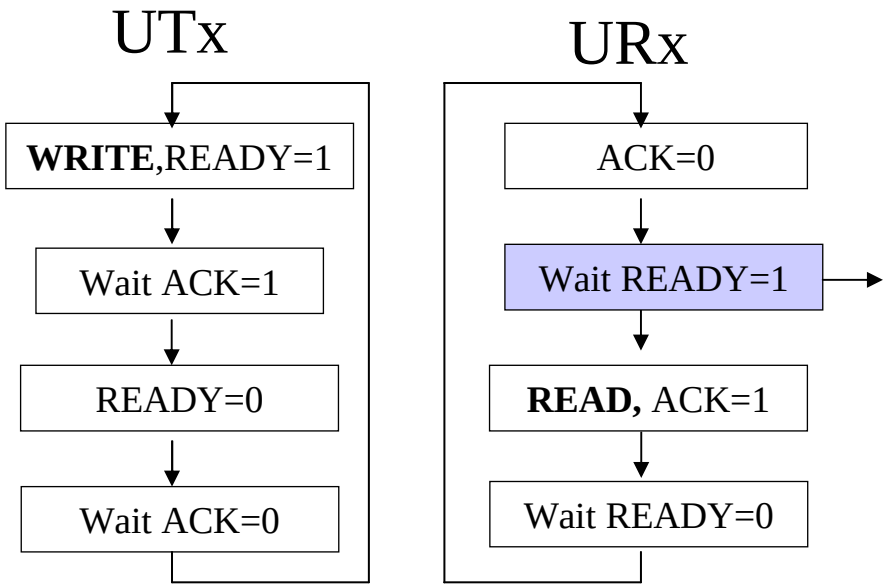


Che succede se si rimuovono le wait?

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

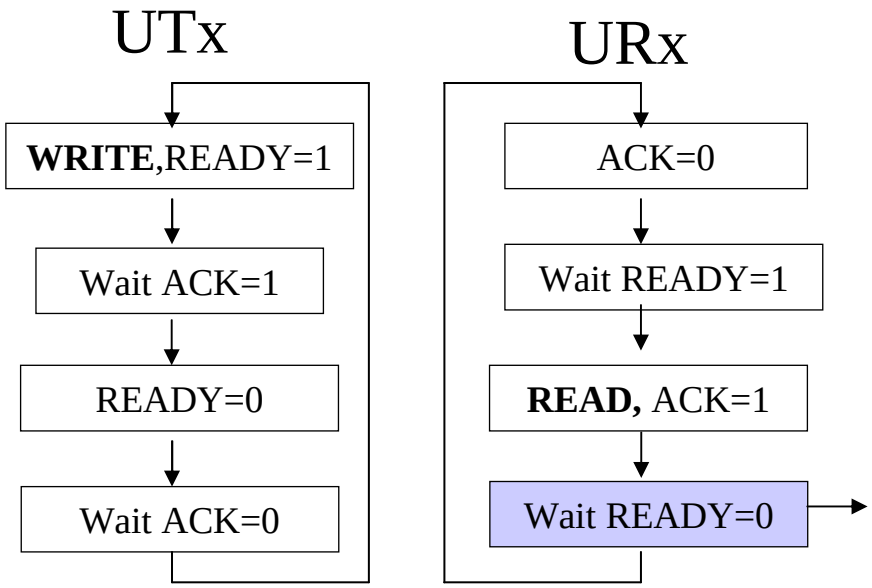


Che succede se si rimuovono le wait?



Potrei leggere prima che la scrittura sia avvenuta!

Che succede se si rimuovono le wait?

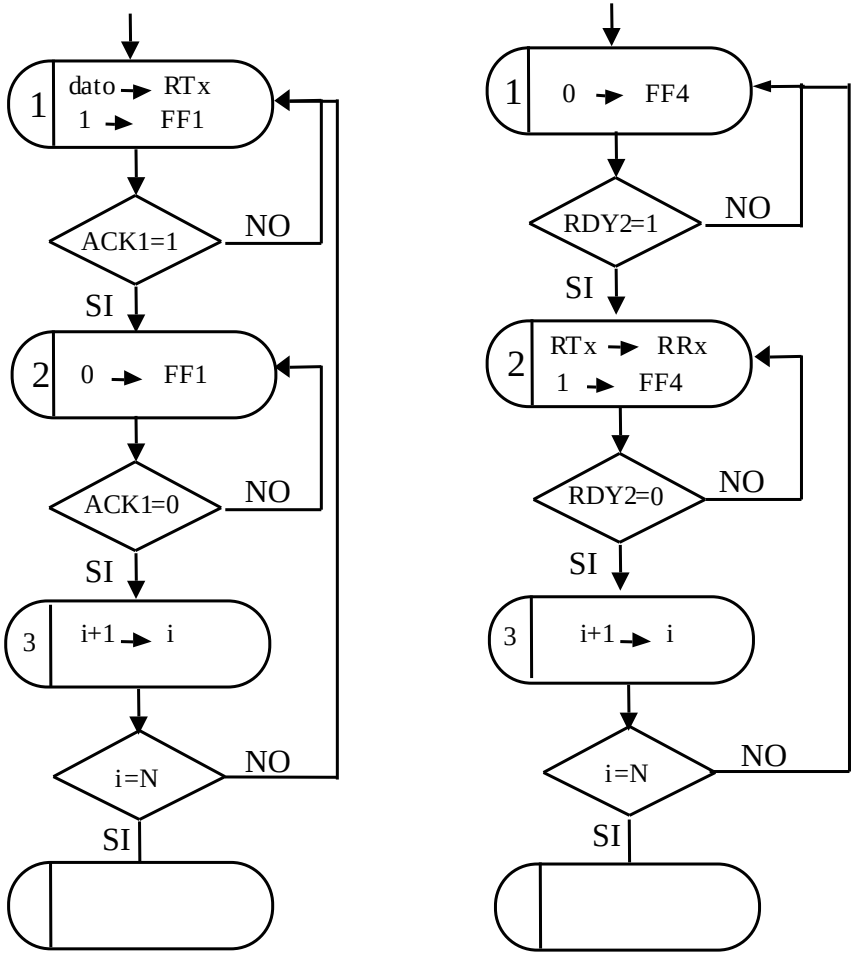


Potrei leggere 2 volte lo stesso valore!

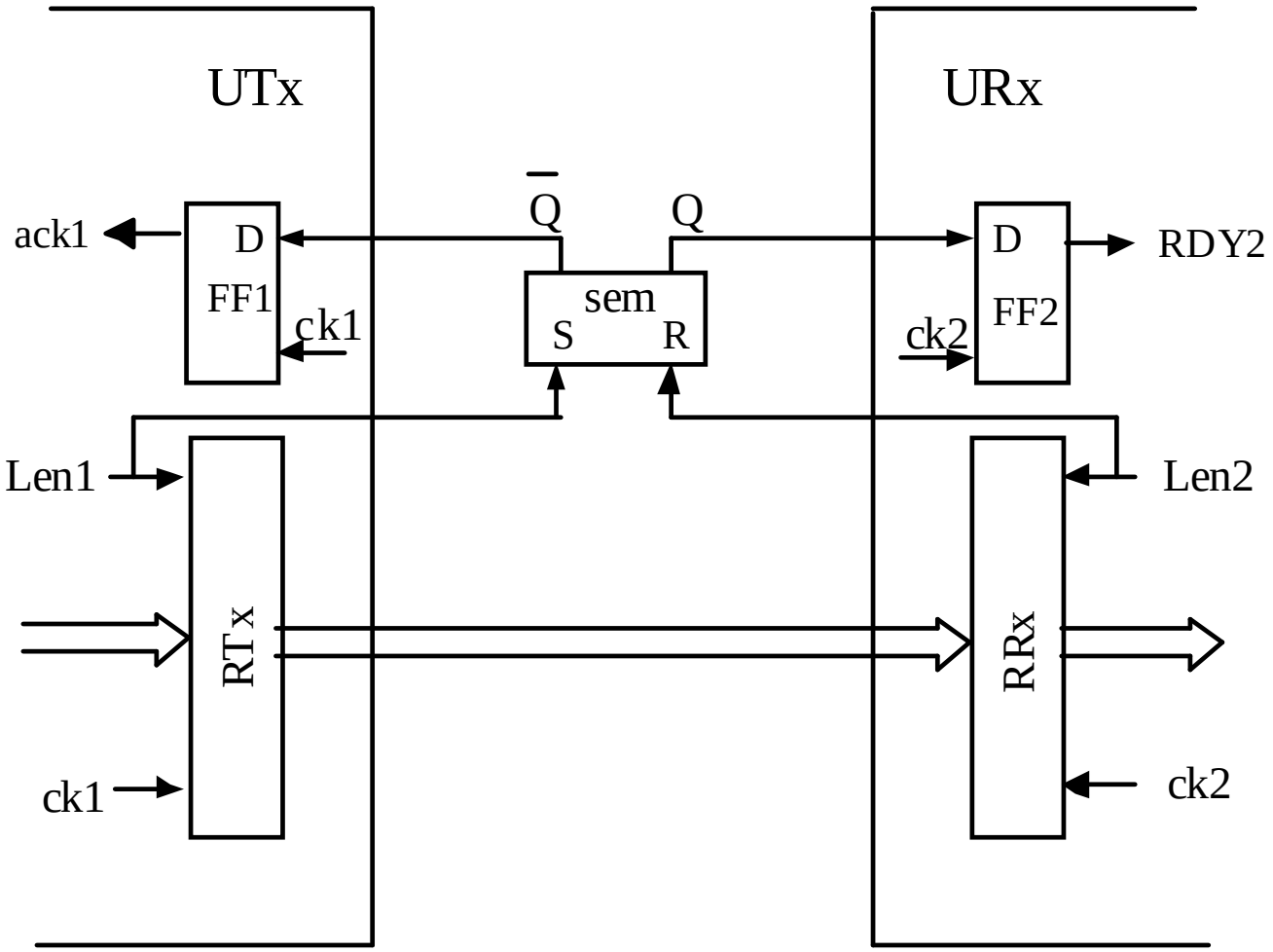
Protocollo di HandShaking

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

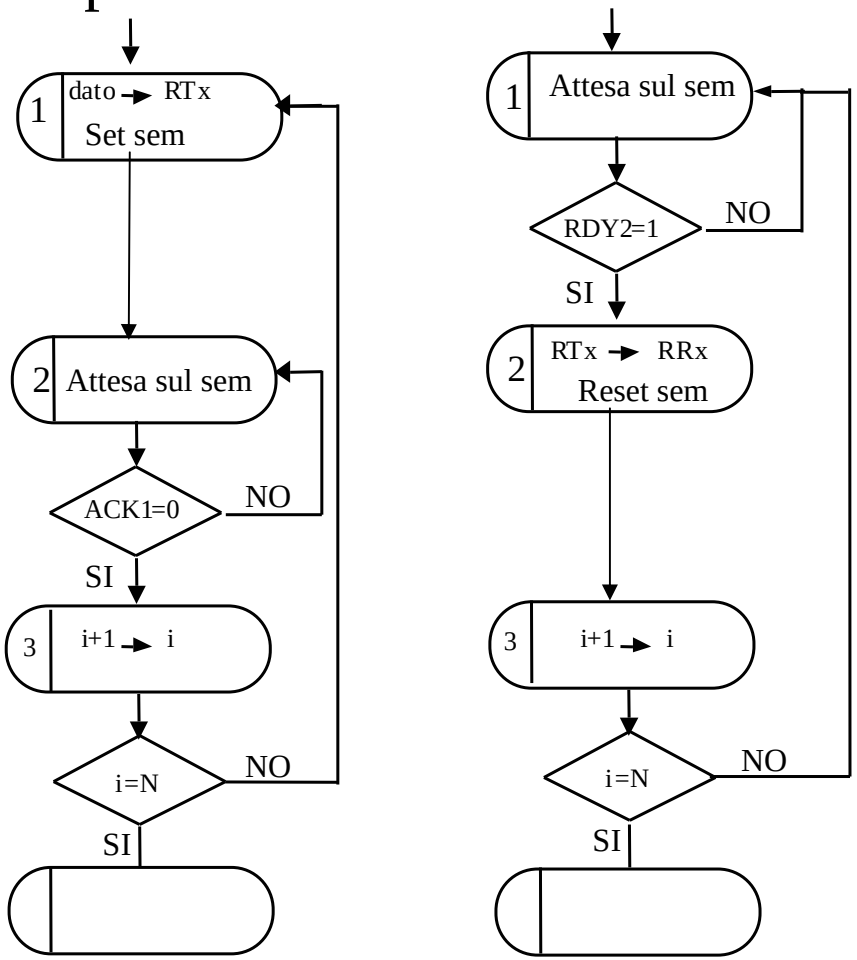
Sequenze di microistruzioni eseguite da UTx e URx durante il protocollo di comunicazione



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



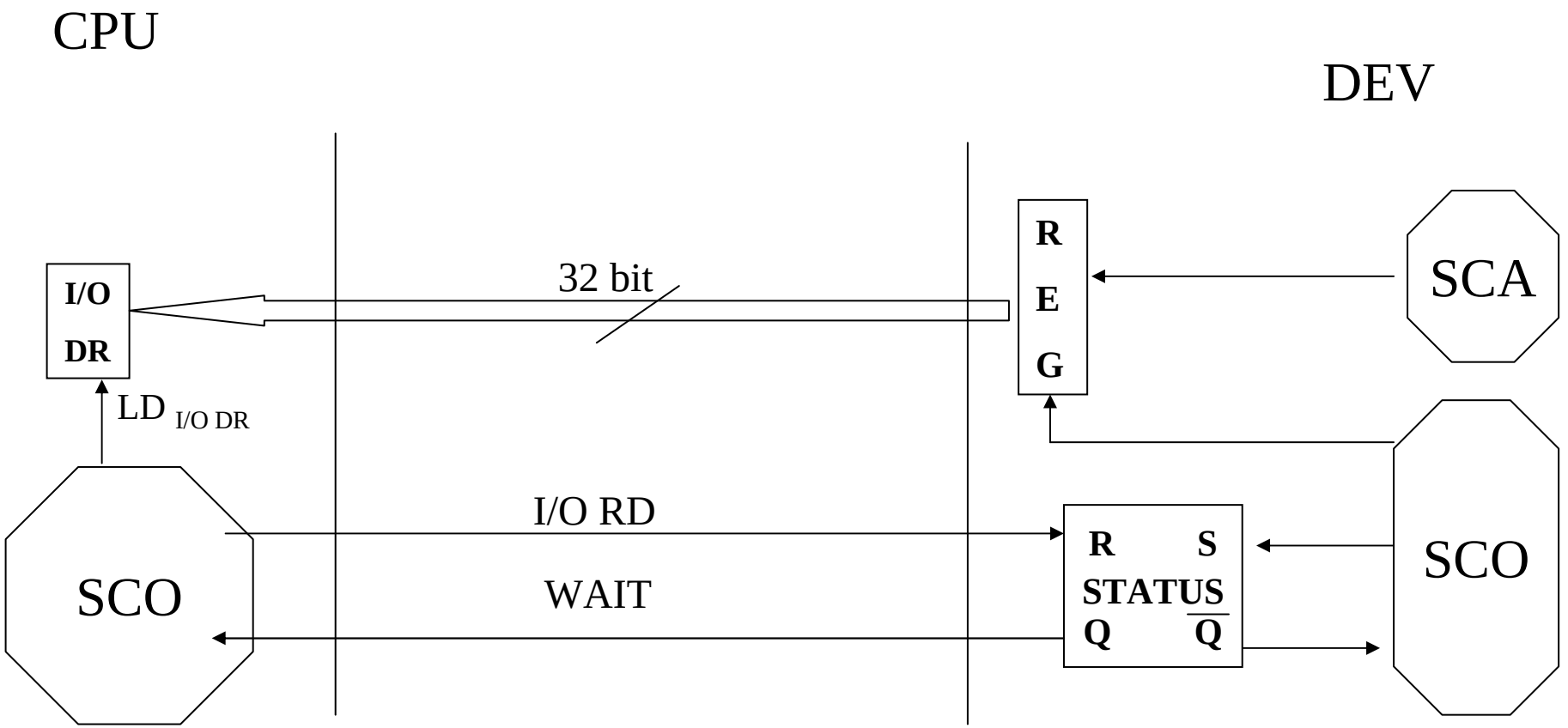
Sequenze di microistruzioni eseguite da UTx e URx durante il protocollo di comunicazione



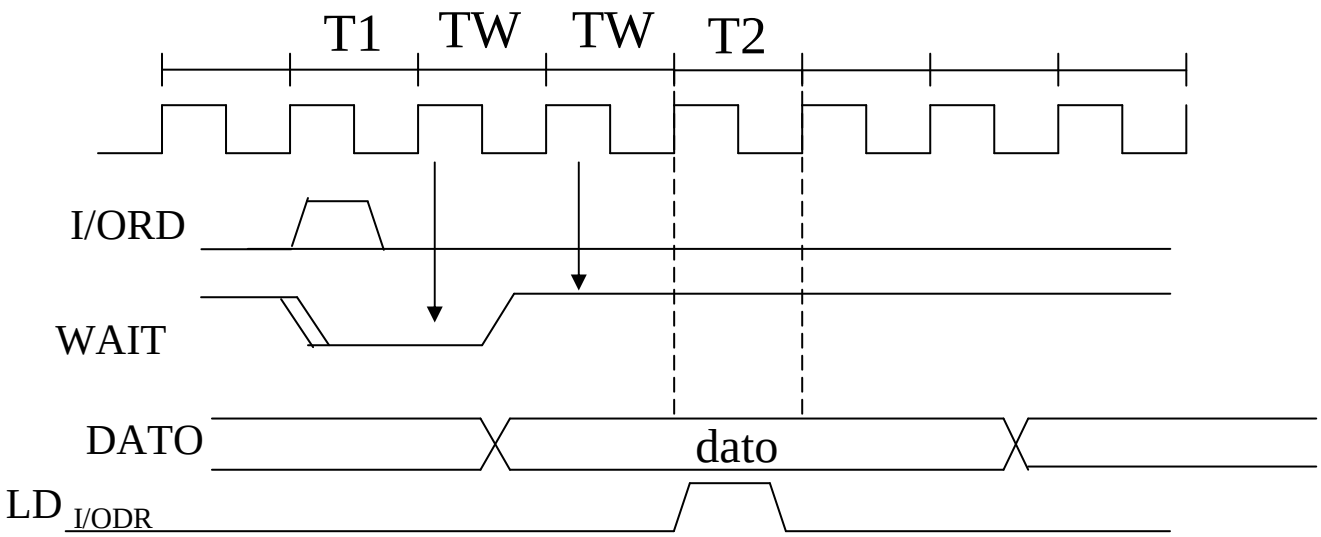
I/O programmato INTERFACCIA DISPOSITIVI di I/O

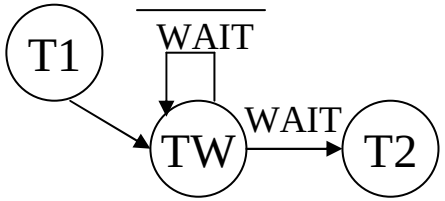
(per supportare protocollo di handshaking, implementato a firmware)

Schema di Interfaccia per l'input tra PD32 e un <u>solo</u> Sistema Digitale Complesso



Temporizzazione dei segnali

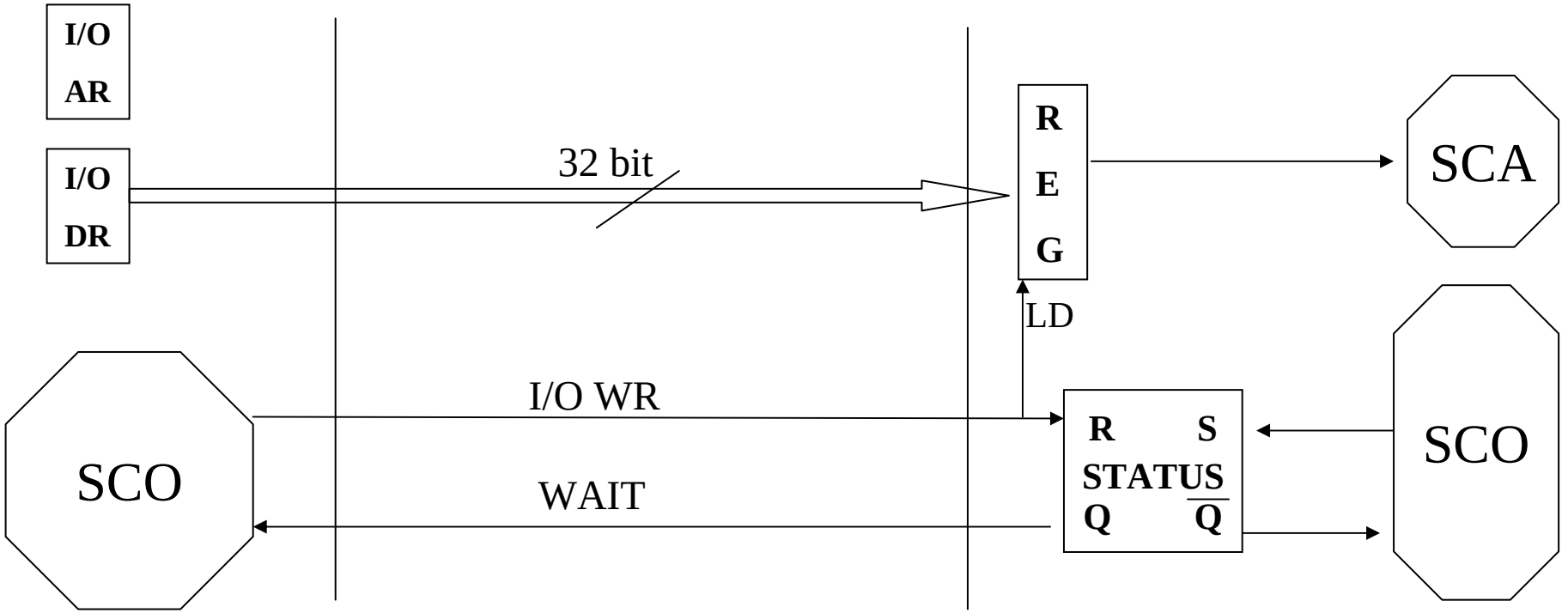




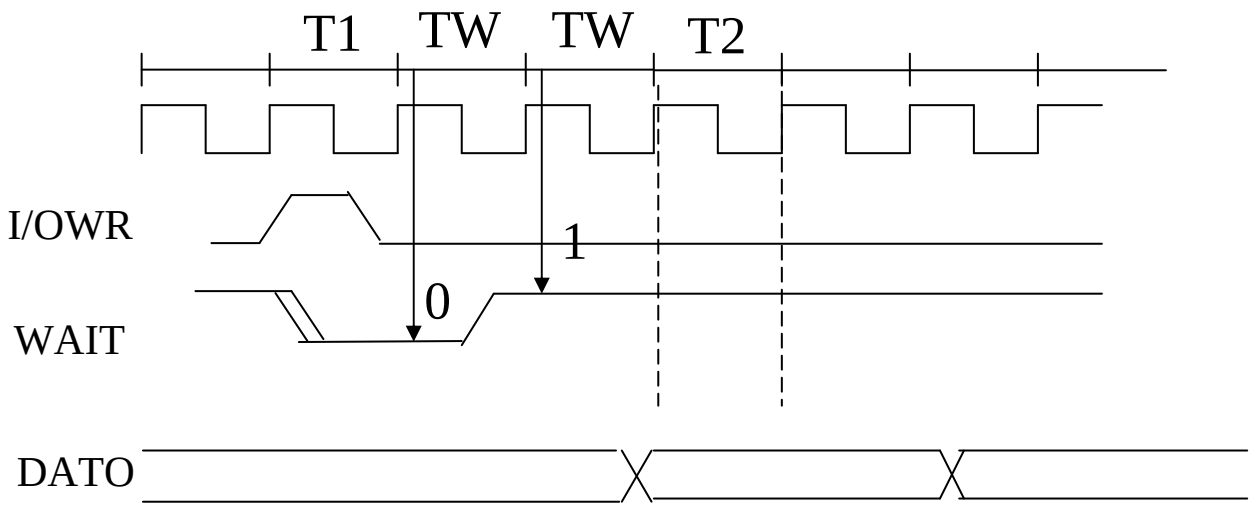
I/O programmato INTERFACCIA DISPOSITIVI DI I/O

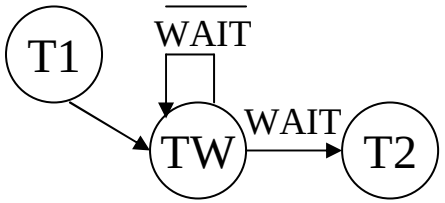
(per supportare protocollo di handshaking, implementato a firmware)

Schema di Interfaccia per l'output tra PD32 e un <u>solo</u> Sistema Digitale Complesso



Temporizzazione dei segnali

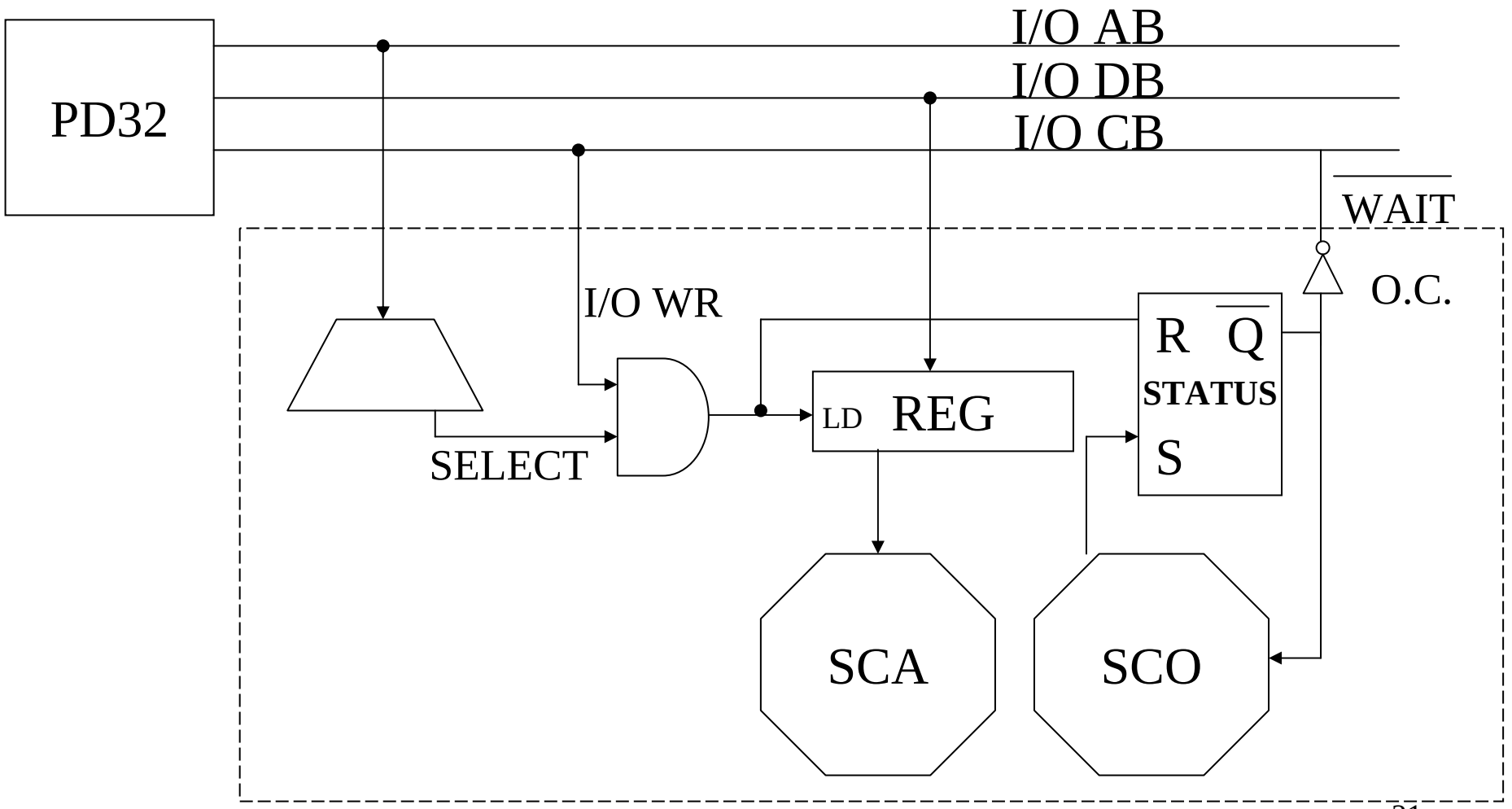




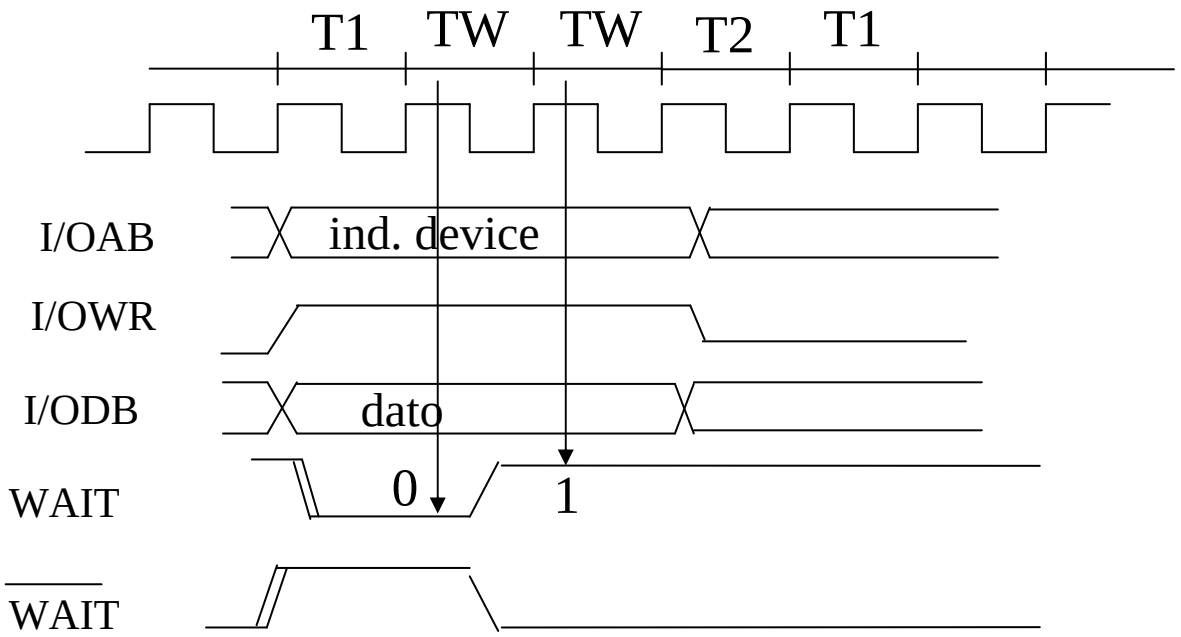
I/O programmato INTERFACCIA DISPOSITIVI DI I/O

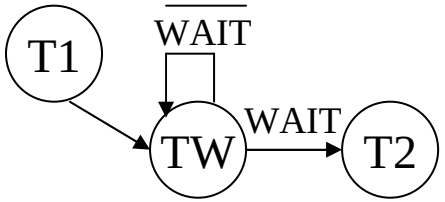
(per supportare protocollo di handshaking, implementato a firmware)

Schema di Interfaccia per l'output tra PD32 e <u>più</u> DISPOSITIVI di I/O



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

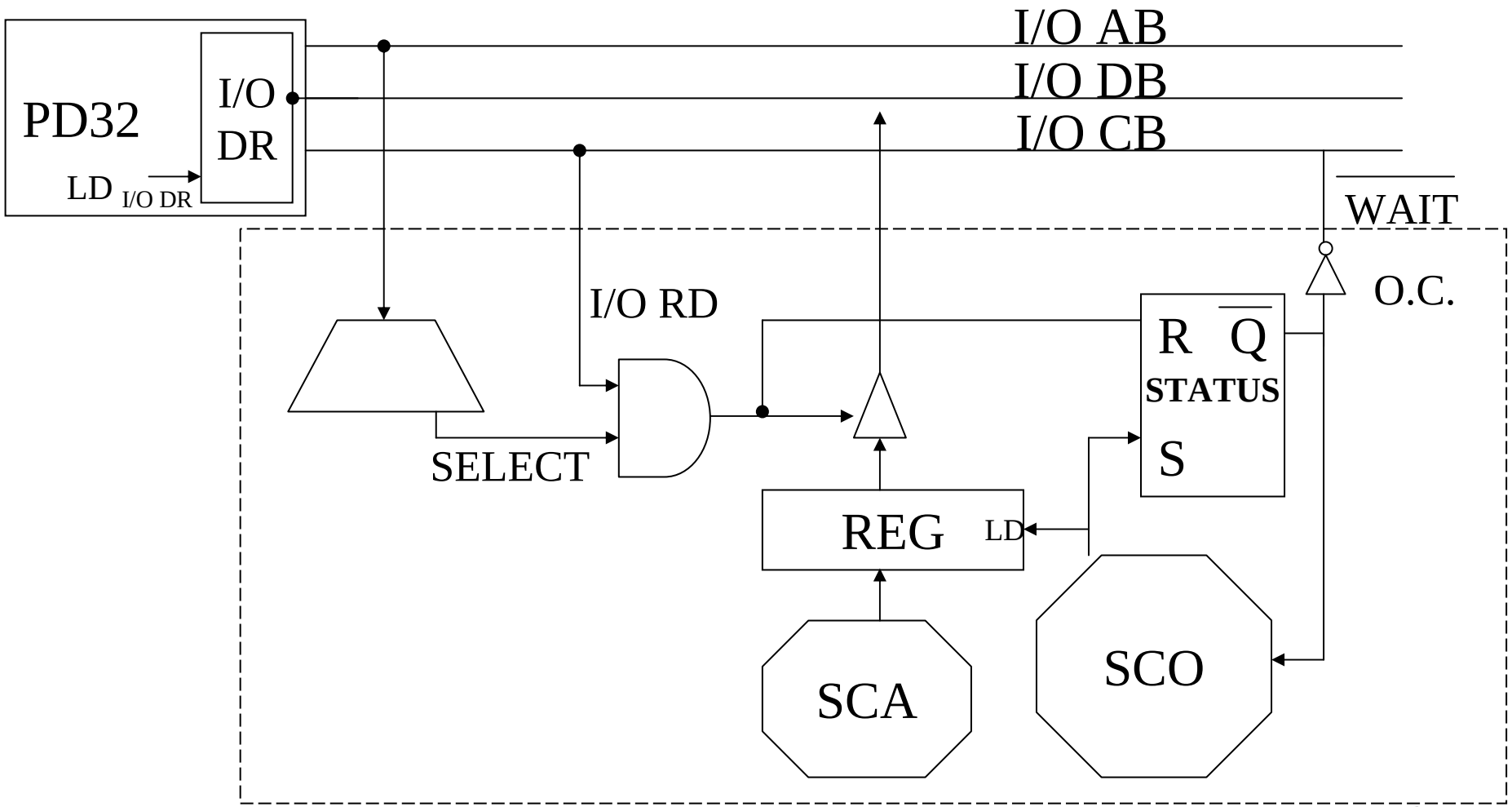




I/O programmato INTERFACCIA DISPOSITIVI DI I/O

(per supportare protocollo di handshaking, implementato a firmware)

Schema di Interfaccia per l'input tra PD32 e <u>più</u> DISPOSITIVI di I/O

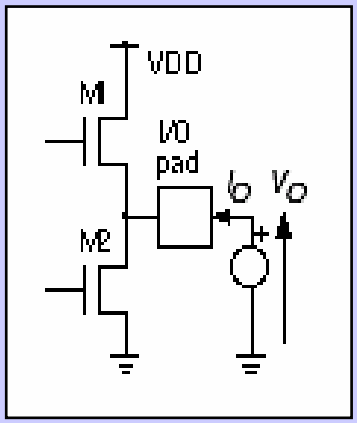


Porte Logiche Open Collector e Connessione Wired-OR

Porte Logiche:
Totem Pole vs Open Collector

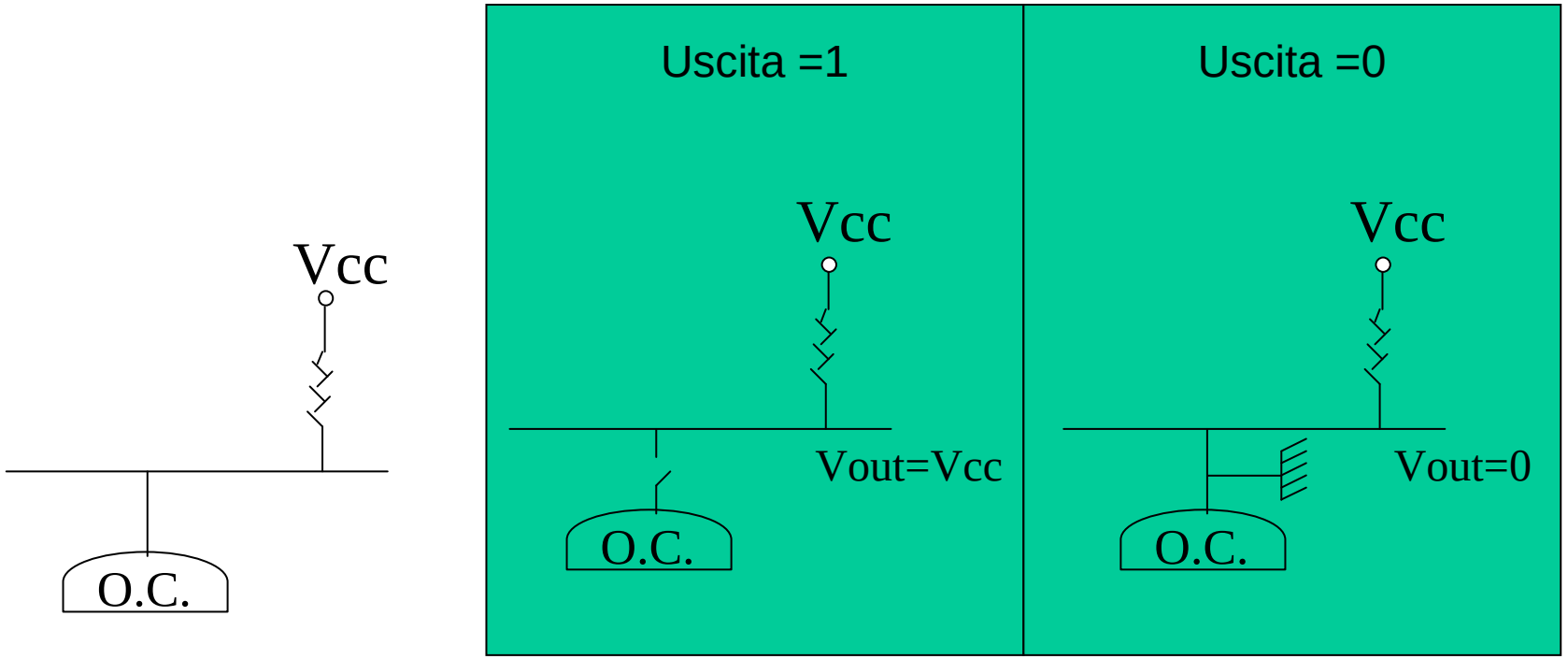
- E’ possibile distinguere due tipologie di porte logiche in funzione dello schema circuitale che le implementa:
  - Totem Pole:
    - cit
      che
    - In caso di
      forza un l
  - Open Collect
    - In caso di u
      impedenza,
    - In caso di usci
      vale 0 (il pin d’usci

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:



Connessione di più porte logiche su uno stesso BUS

- **PROBLEMA**: Non è possibile connettere più porte logiche Totem Pole sullo stesso BUS, onde evitare conflitti dovuti alla presenza di stati logici diversi su porte logiche diverse.
- **Soluzioni**:
  - Utilizzare buffer three states oppurtanamente pilotati per garantire che solo una porta logica sia effettivamente connessa al bus in ogni istante.
  - Utilizzare porte logiche OPEN COLLECTOR +una connessione di tipo wired-or.

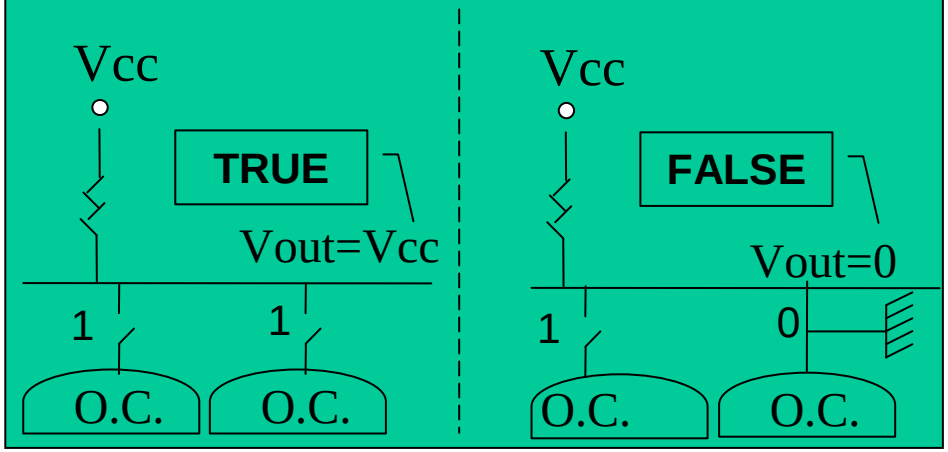
WIRED OR / WIRED AND

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).

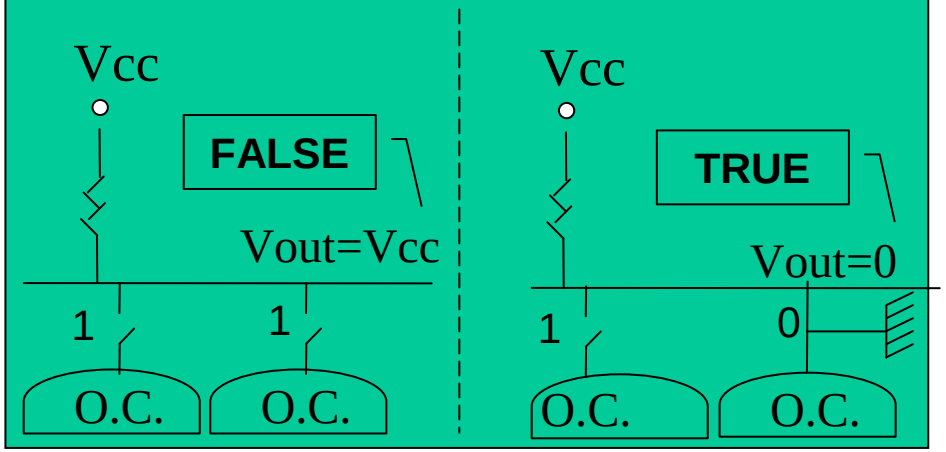
AND DELL'USCITA DELLE SINGOLE PORTE



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



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

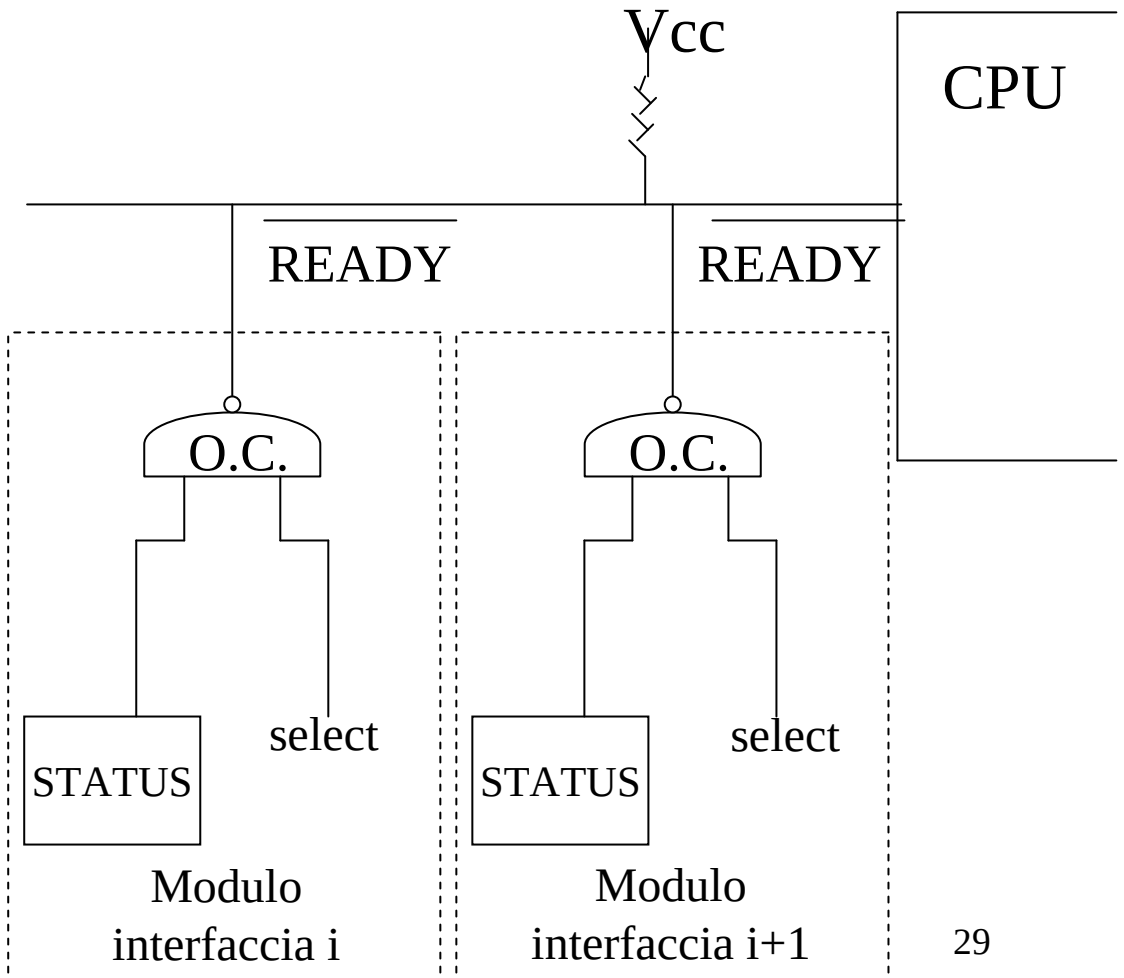
A) Se nessuna interfaccia ha attivo il segnale di select, tutti i NAND O.C. vanno in alta impedenza e $\overline{\text{READY}}$=1 (false).

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

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

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

Questo ci consente di evitare conflitti!



ISTRUZIONE DI OUTPUT

OUTs Rx, Device

(sposta nel registro di interfaccia del dispositivo di indirizzo Device uno o due o quattro bytes memorizzati nel registro Rx, naturalmente il numero di bytes da spostare dipende dal valore di “s”)

ISTRUZIONE DI INPUT

INs Device, Rx

(sposta nel registro Rx uno o due o quattro bytes memorizzati nel registro di interfaccia del dispositivo di indirizzo Device, naturalmente il numero di bytes da spostare dipende dal valore di “s”)

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)

Necessità di istruzioni atte alla emulazione del test sulla variabile di condizione WAIT (negata) – uso di una nuova variabile di condizione “READY” - e di una istruzione per avvertire SCO del dispositivo di I/O che il processore intende colloquiare con la periferica.

JR address (se $\overline{\text{READY}} = 0$ allora salta all’esecuzione dell’istruzione il cui indirizzo di memoria è “address”)

JNR address (se $\overline{\text{READY}} = 1$ allora salta all’esecuzione dell’istruzione il cui indirizzo di memoria è “address”)

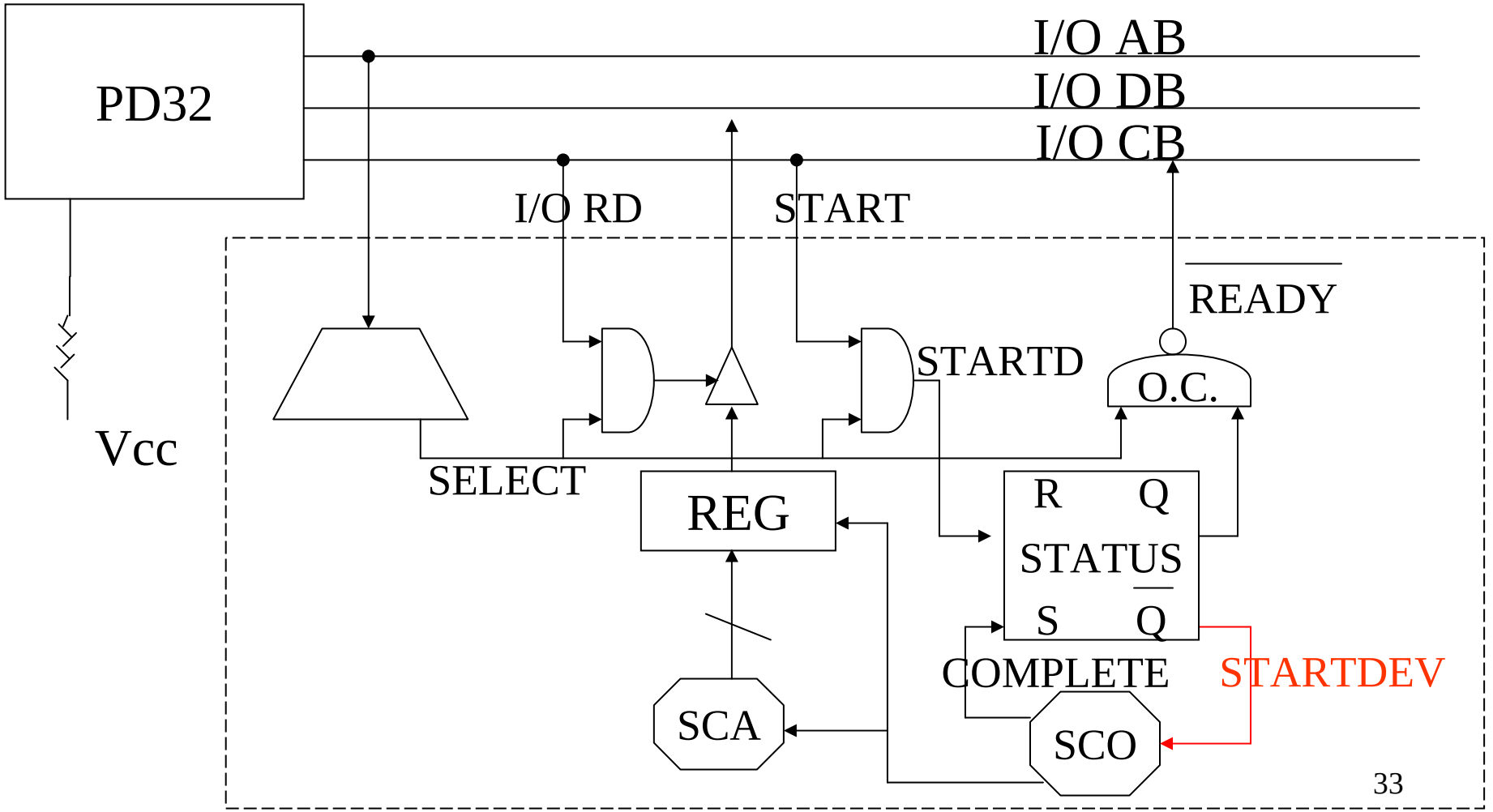
START device (avverti – asserendo ad 1 il segnale di controllo START – la periferica “device” che ci sarà un’interazione)

Notare che per eliminare gli effetti della variabile di condizione $\overline{\text{WAIT}}$

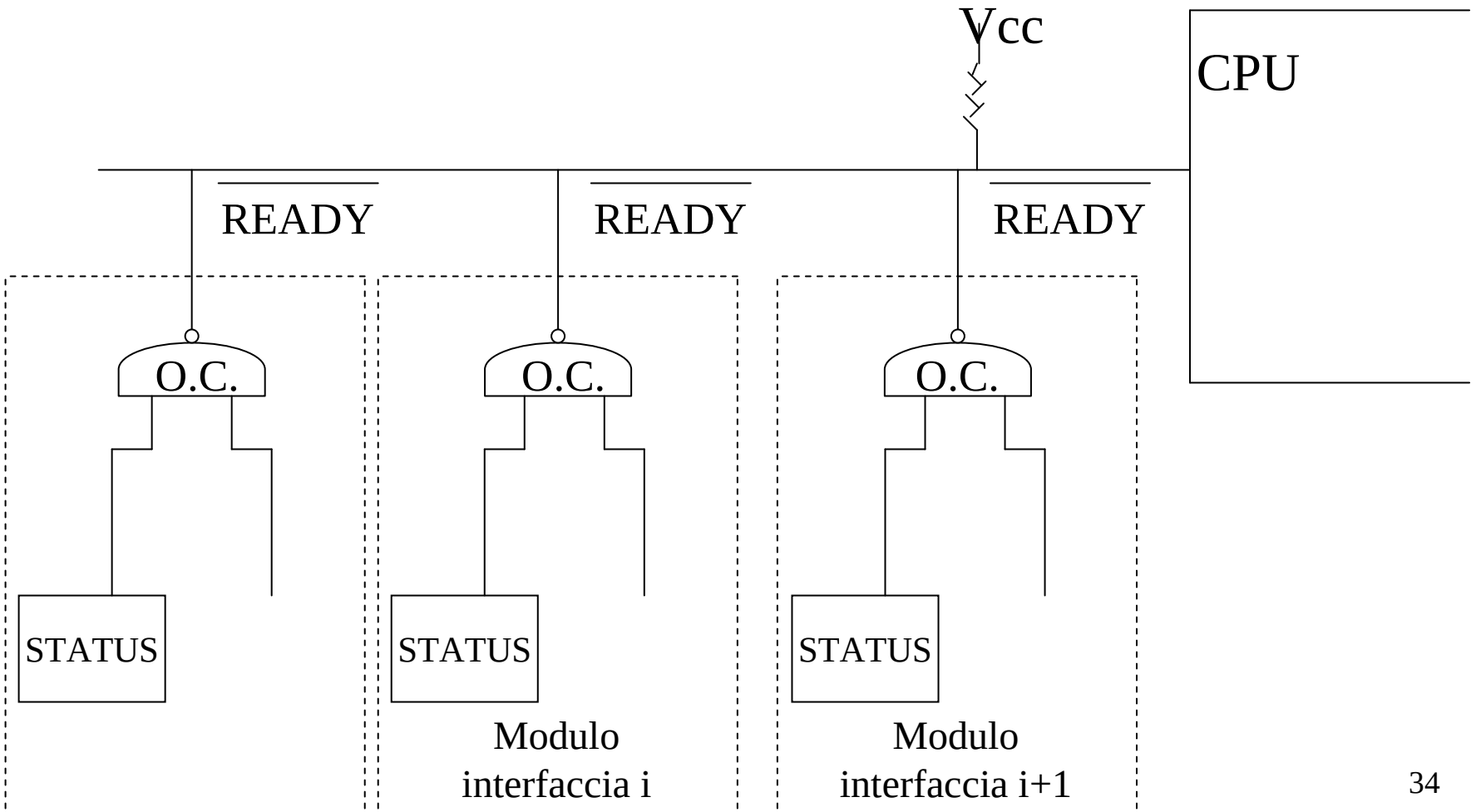
è sufficiente mettere a massa (a zero) il collegamento (pin) del processore relativo a tale variabile. Notare che in questo caso durante l’esecuzione di una istruzione di IN o di OUT il processore passa una volta solo nello stato WAIT (vedere lucidi precedenti).

I/O programmato – INTERFACCIA di INPUT

PROTOCOLLO DI HANDSHAKING IMPLEMENTATO A SOFTWARE



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



Protocollo di Input

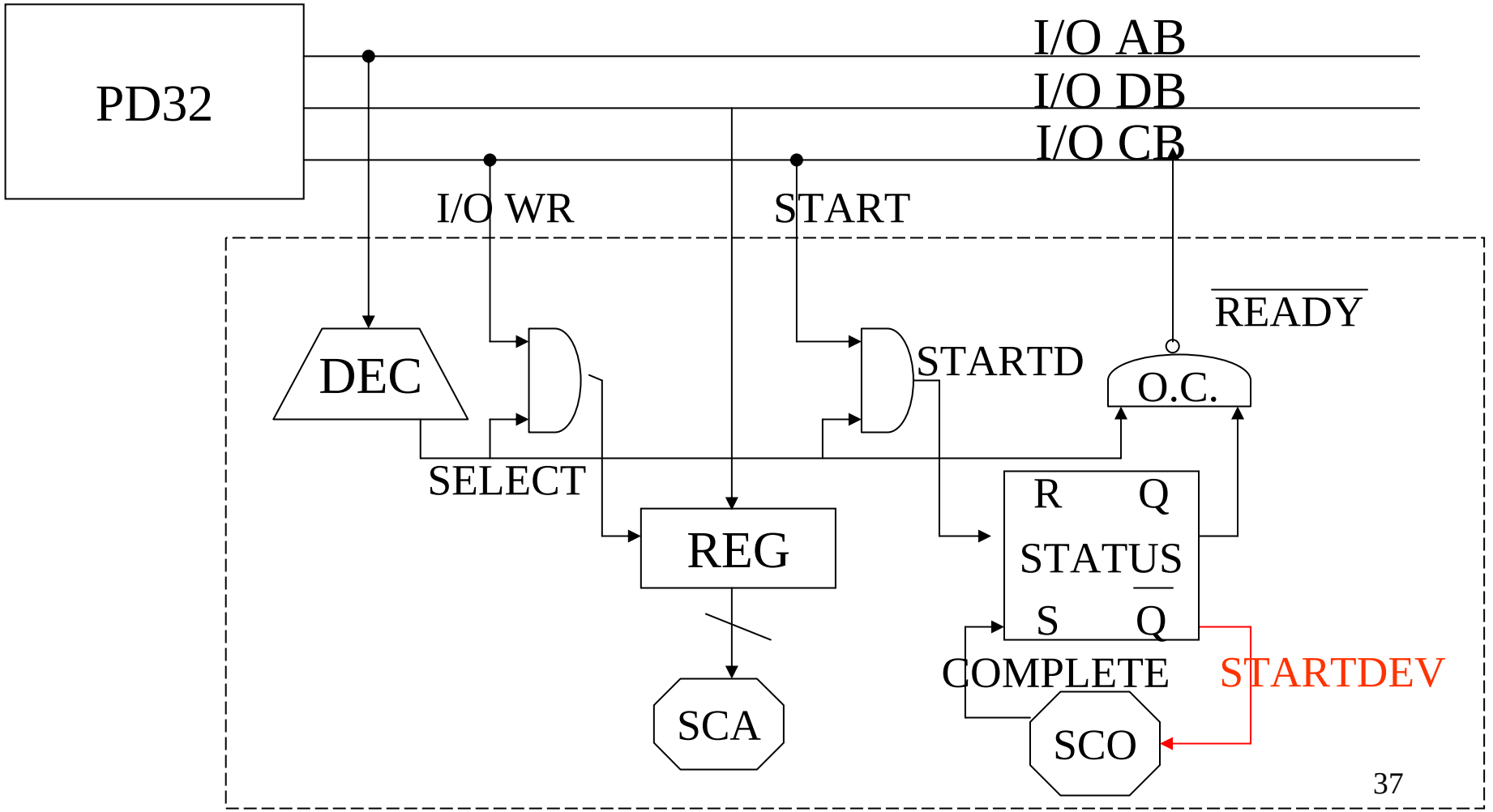
1. Il processore invia sull'I/O Address bus l'indirizzo del dispositivo e ne esamina lo stato tramite la linea di controllo $\overline{\text{READY}}$.
2. Se il dispositivo non è pronto ($\overline{\text{READY}} = 1$) il processore deve attendere e tornare al punto 1 o saltare ad un'altra istruzione.
3. Se $\overline{\text{READY}} = 0$ il processore avverte il dispositivo che vuole un dato da lui (seleziona il dispositivo tramite le linee indirizzi e invia il segnale START). START resetta il flip-flop STATUS e in tale stato rimane per tutta la durata delle operazioni di produzione del dato da parte del dispositivo.
4. Quando il dato è stato prodotto ed è disponibile in REG, il dispositivo genera il segnale COMPLETE, settando STATUS ($\overline{\text{READY}}=0$).
5. Nel frattempo il processore, in attesa del dato, esamina lo stato del F/F campionando il segnale READY.
6. Se $\overline{\text{READY}}= 1$ il processore deve attendere e tornare al punto 5.
7. Se $\overline{\text{READY}}= 0$ il processore esegue una istruzione id 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).

Programma assembler (input)

```
Aspetta1:    JNR DeviceIN, Aspetta 1

             START DeviceIN

Aspetta2:    JNR DeviceIN, Aspetta2

             INB DeviceIN, R0
```

I/O programmato- INTERFACCIA di OUTPUT

PROTOCOLLO DI HANDSHAKING IMPLEMENTATO A SOFTWARE



Protocollo di Output

1. Il processore invia sull'I/O Address bus l'indirizzo del dispositivo e ne esamina lo stato tramite la linea di controllo $\overline{\text{READY}}$.
2. Se il dispositivo non è pronto ($\overline{\text{READY}} = 1$) il processore deve attendere e tornare al punto 1 o saltare ad un'altra istruzione.
3. Se $\overline{\text{READY}}=0$ 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).
4. Il processore avverte il dispositivo che gli ha trasferito un dato (seleziona il dispositivo tramite le linee indirizzi e invia il segnale START). START resetta il flip-flop STATUS e in tale stato rimane per tutta la durata delle operazioni di consumo del dato da parte del dispositivo. Quando il dato è stato letto da REG, il dispositivo genera il segnale COMPLETE, settando STATUS ($\overline{\text{READY}}=0$).
5. Nel frattempo il processore, in attesa, esamina lo stato del F/F campionando il segnale $\overline{\text{READY}}$.
6. Se $\overline{\text{READY}}= 1$ il processore deve attendere e tornare al punto 5.
7. Se $\overline{\text{READY}}= 0$ il processore può eseguire un'altra istruzione.

Programma assembler (output)

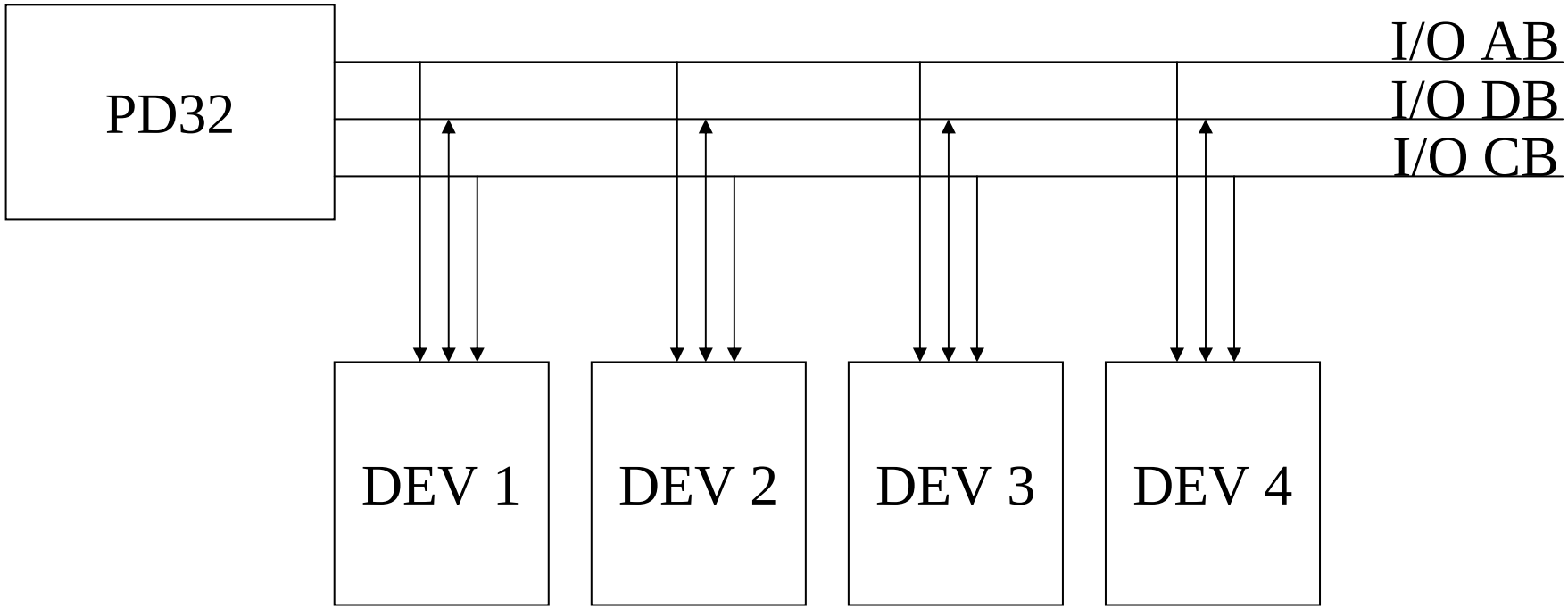
```
Aspetta1:    JNR DeviceOUT, Aspetta 1

             OUT R0, DeviceOUT

             START DeviceOUT

Aspetta2:    JNR DeviceOUT, Aspetta2
```

I/O programmato
MODALITA' BUSY WAITING

```
…
MOVL #100, R0     ; numero di dati da acquisire
MOVL #DATI, R1 ; ind.dell'area di memoria
JSR     IN_AD1
…
IN_AD1: PUSH R0             ; salv. registri usati
        PUSH R1
        PUSH R2
IN_1:   JNR AD1, IN_1       ; attende che AD1 sia pronto
IN_2:   START AD1           ;avvia l'acquisizione di un dato
IN_3:   JNR AD1, IN_3       ; attende che il dato sia stato prodotto
        INW AD1, R2         ; prelievo del dato e….
        MOVW R2, (R1)+ ; … suo trasferimento in memoria
        SUBL  #1, R0        ; decremento del contatore
        JNZ IN_2            ; acquisizione di un altro dato se non si è azzerato
                            ; il contatore
        POP R2              ; ripristino dei registri usati
        POP R1              ;
        POP R0              ;
        RET                 ; ritorno al programma chiamante
```

Polling

(verifica circolare se ogni DEVICE è pronto ad interagire)



Polling

```
…
MOVL #100, R0          ; numero di dati da acquisire
MOVL #DATI, R1         ; ind.dell'area di memoria
JSR    IN_AD1
…
IN_AD1: PUSH R0        ; salv. registri usati
            PUSH R1
            PUSH R2
POLL:       JR AD1, IN_1          ; attende che AD1 sia pronto
            JR AD2, IN_2
            JR AD3, IN_3
            JR AD4, IN_4
            JMP POLL

IN_i:       START ADi             ; avvia l'acquisizione di un dato
WAIT:       JNR ADi, WAIT         ; attende che il dato sia stato prodotto
            INW ADi, R2           ; prelievo del dato e….
            MOVW R2, (R1)+ ; … suo trasferimento in memoria
            SUBL  #1, R0          ; decremento del contatore
            JNZ WAIT  ; acquisizione di un altro dato se non si è azzerato
                                  ; il contatore
            POP R2                ; ripristino dei registri usati
            POP R1                ;
            POP R0                ;
            RET                   ; ritorno al programma chiamante
```

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

1) salvare lo stato del processo in esecuzione;

2) identificare il programma di servizio relativo all'interruzione

3) eseguire il programma di servizio;

4) riprendere le attività lasciate in sospeso.

Tecniche di identificazione del programma di servizio relativo all'interruzione

(identificazione della sorgente dell'interruzione)

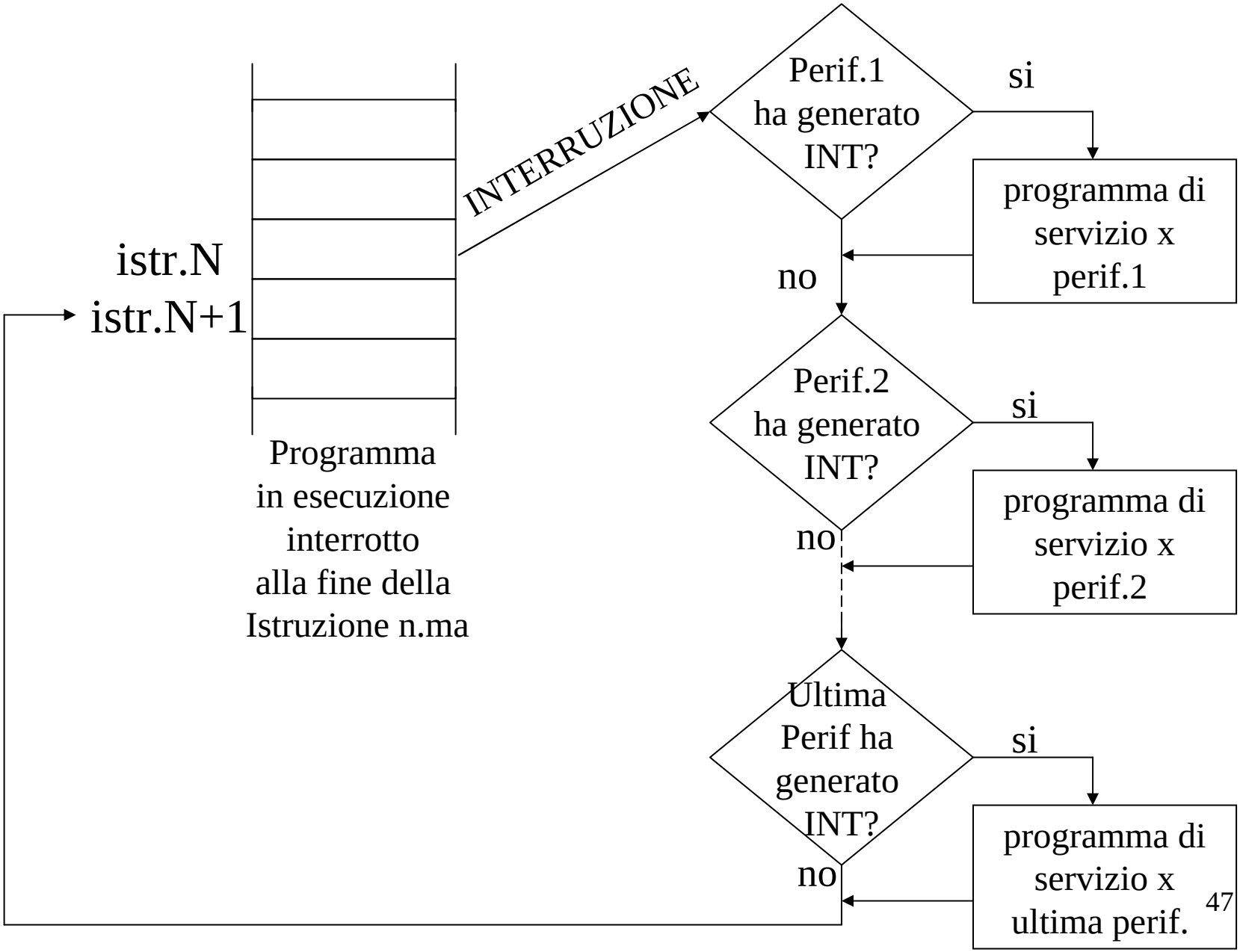
Polling

Polling a multilivello

Vettorizzata

Vettorizzata a multilivello

Polling


istr.N
istr.N+1
Programma
in esecuzione
interrotto
alla fine della
Istruzione n.ma
INTERRUZIONE
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

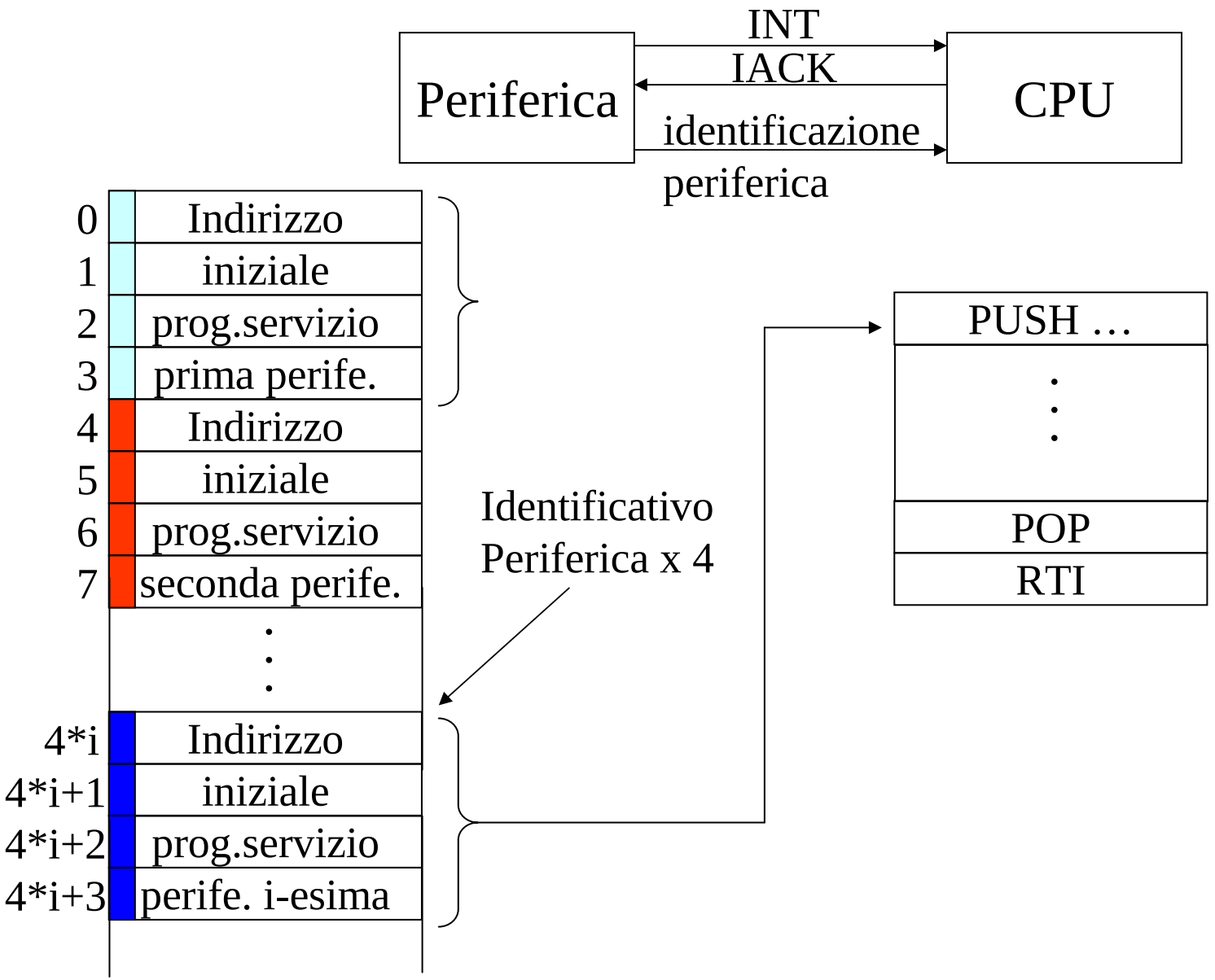
Routine di Polling per il riconoscimento delle interruzioni

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

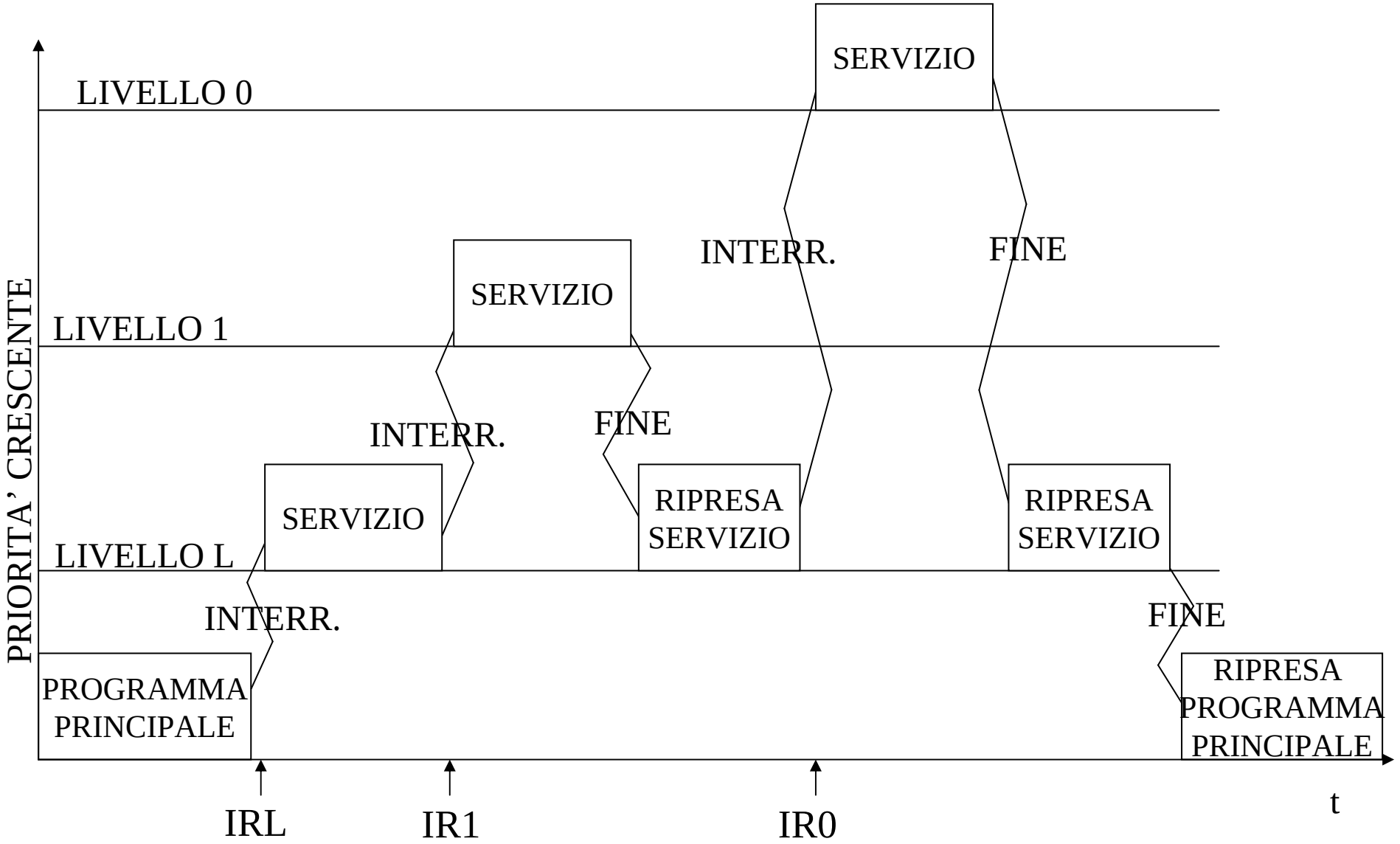
- Richieste multiple vengono servite in ordine di interrogazione
- Tempo di CPU non minimo per il riconoscimento

N.B “JP” non è una istruzione PD32

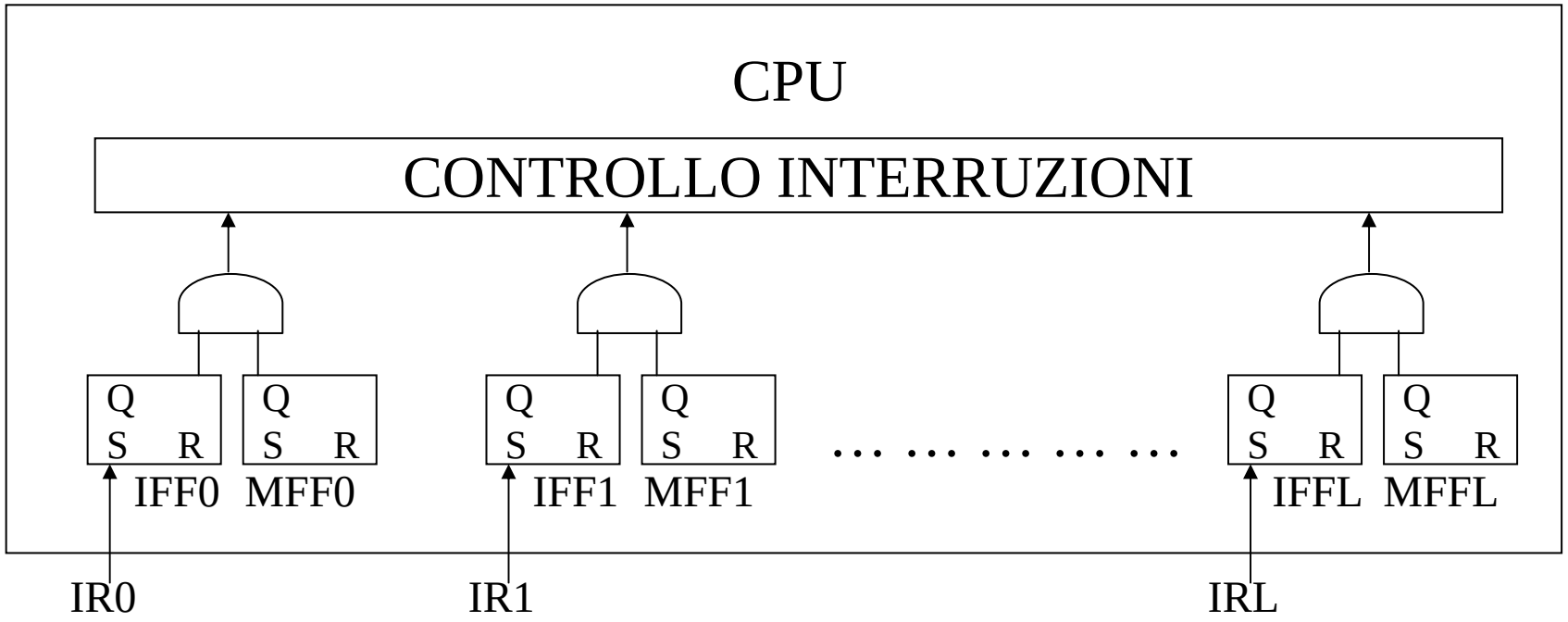
Meccanismo interruzioni vettorizzate



Meccanismo di interruzione multilivello (a polling o vettorizzate)



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

Gestione delle richieste di interruzione PD32

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 assembler CLRI e SETI.

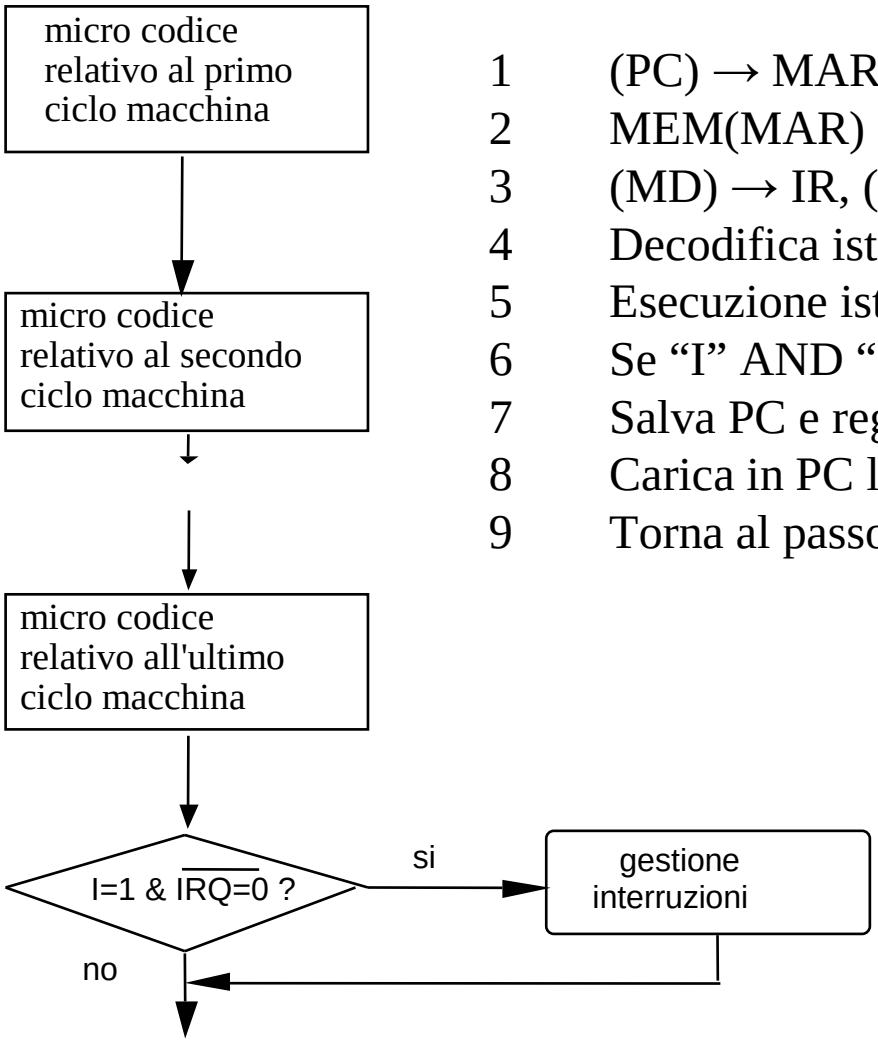
Verifica richiesta delle interruzioni

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 PD32



1	(PC) → MAR	--- fetch
2	MEM(MAR) → MD	--- fetch
3	(MD) → IR, (PC)+1 → PC	--- fetch
4	Decodifica istruzione	
5	Esecuzione istruzione	
6	Se “I” AND “$\overline{\text{IRQ}}$” = 0 allora vai a 7 altrimenti a 9	
7	Salva PC e registro dei flag nella pila puntata dallo SP	
8	Carica in PC l'indirizzo iniziale del programma di servizio	
9	Torna al passo 1 fino a che trovi una istruzione HALT	

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).

Salvataggio dello Stato

Organizzazione a pila (stack di sistema- gestione LIFO) della memoria in cui andare memorizzare i contenuti dei registri (vedere gestione delle subroutine).

R7 è visto anche come STACK POINTER dello stack di sistema.

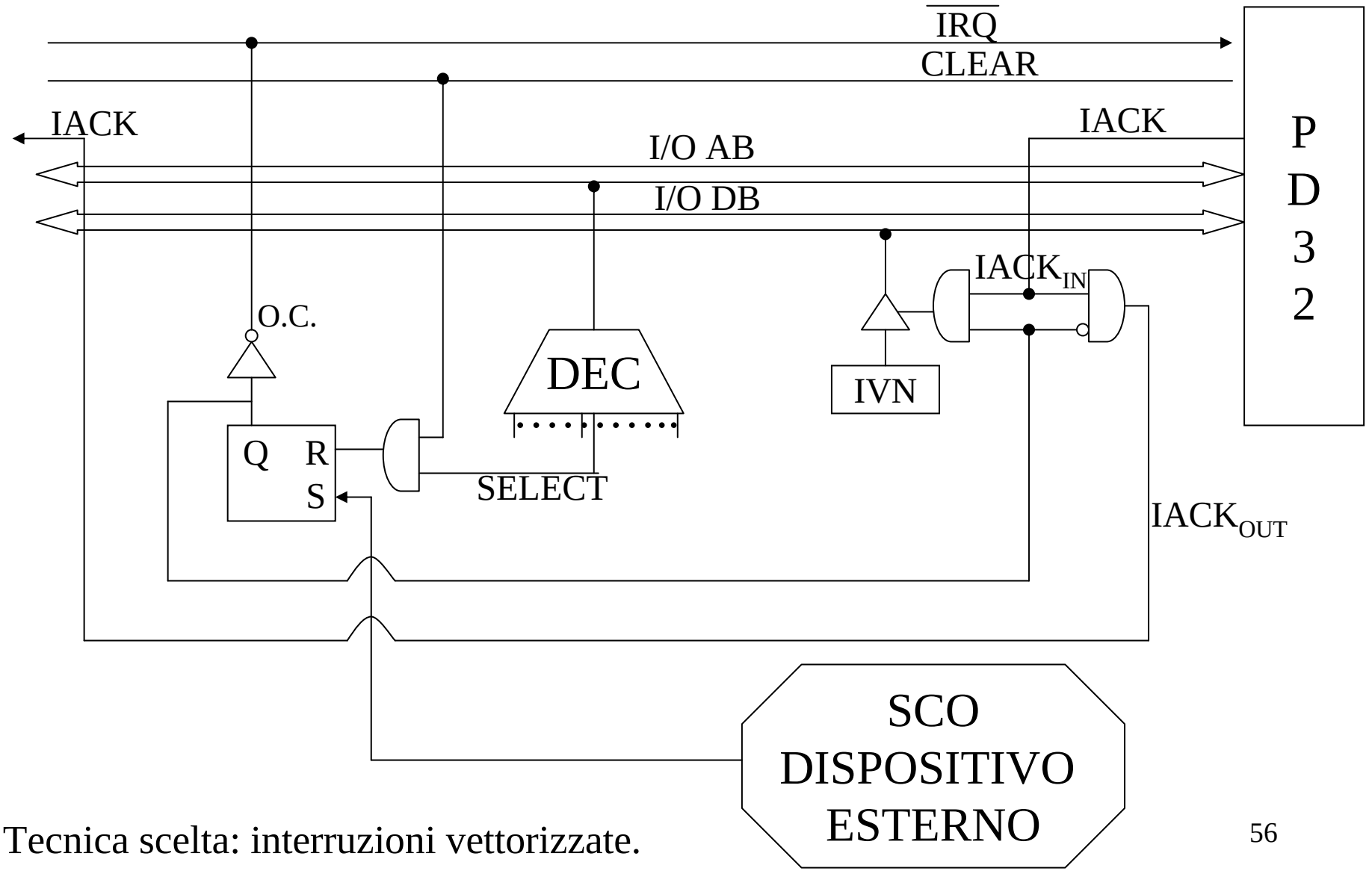
Possibilità di memorizzazione:

via firmware tutti i registri

via firmware solo PC e SR, a software solo quelli che verranno effettivamente modificati.

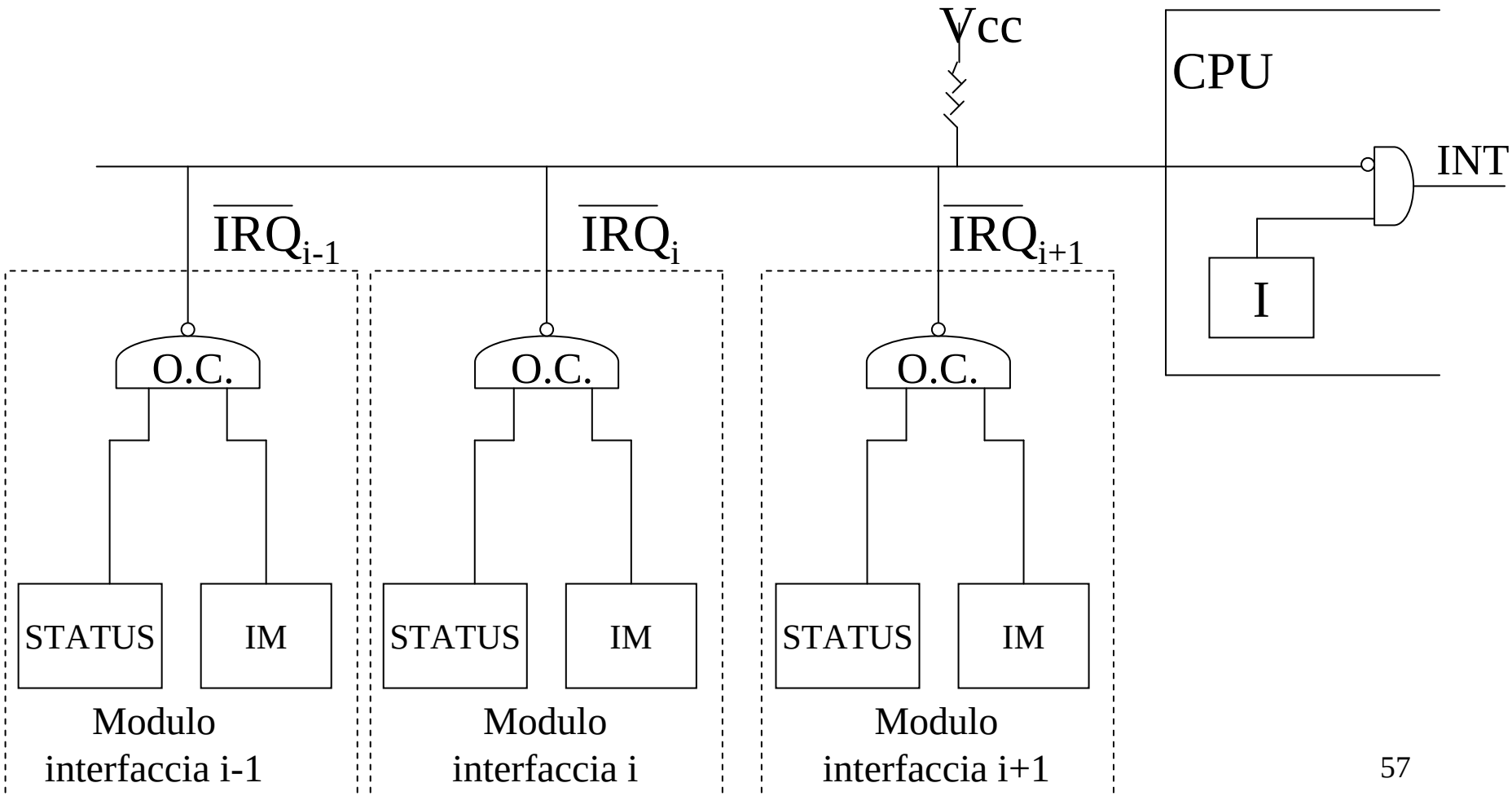
Nel PD32 si è optato per la seconda soluzione.

Identificazione programma di servizio



Tecnica scelta: interruzioni vettorizzate.

Connesione, 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: Ri) che la routine di servizio modificherà tramite esecuzione di istruzioni tipo PUSH.

Esecuzione programma.

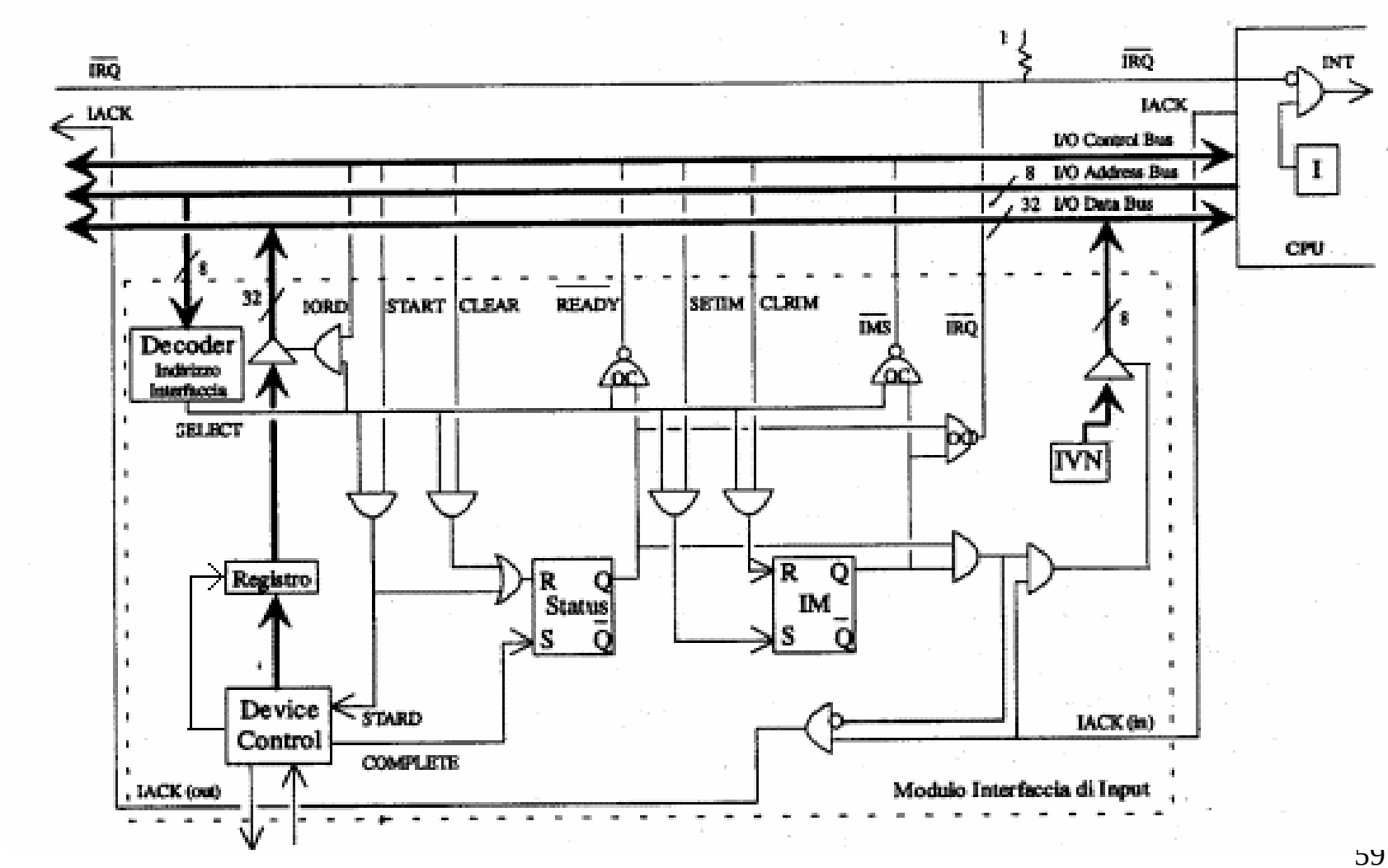
Ripristino del contenuto dei registri salvati nello stack, tramite POP (tante POP per quante PUSH effettuate precedentemente).

Rimuovere la richiesta di Interrupt : START / CLEAR/ CLEARIM

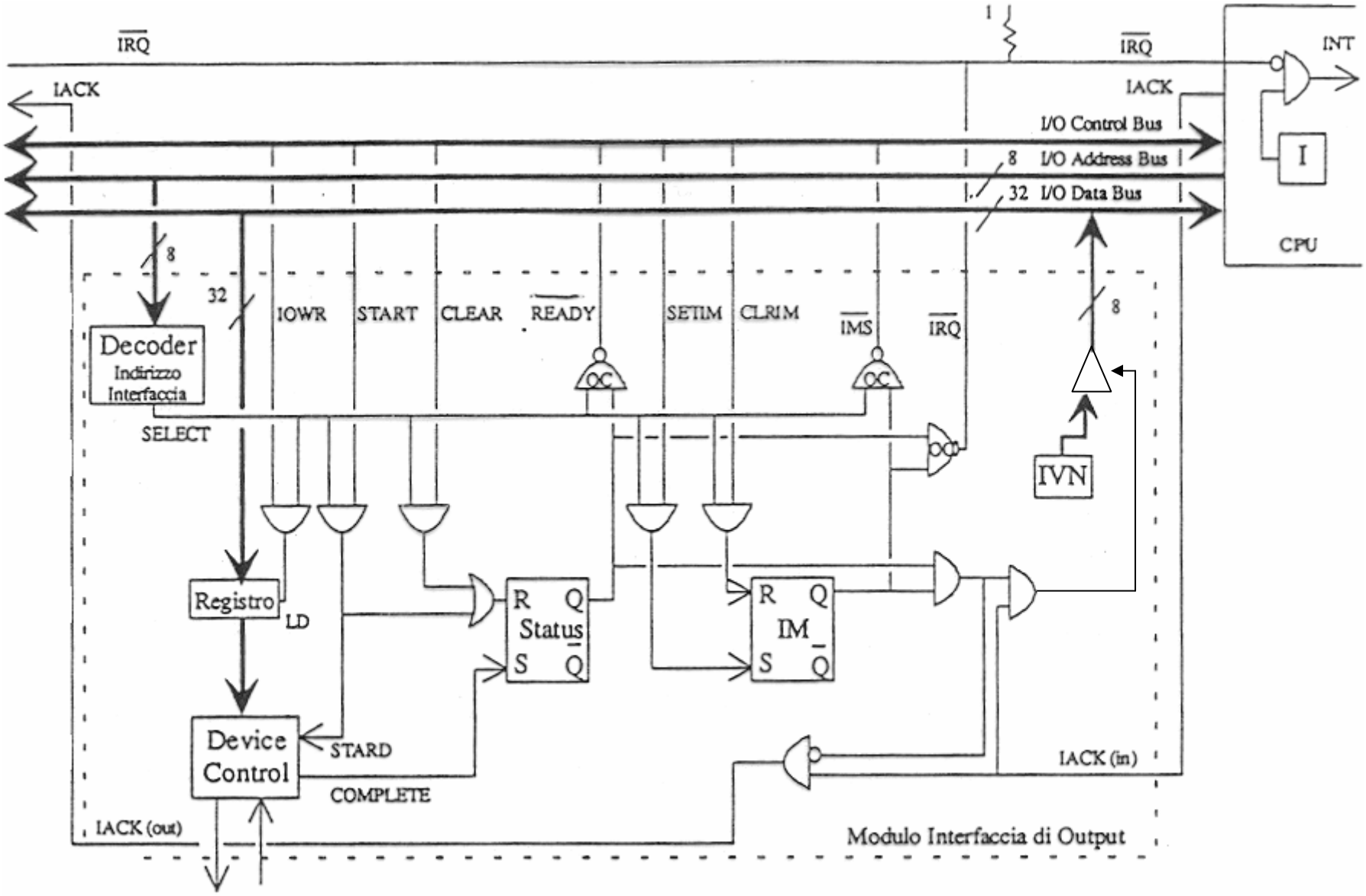
Esecuzione della RTI (**R**e**T**urn from **I**nterrupt), che ripristina nello SR e nello PC i valori memorizzati nello stack di sistema (equivalente a due istruzioni POP).

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

Interfaccia di Input e sua connessione con il PD32



Interfaccia di Output e sua connessione con il PD32



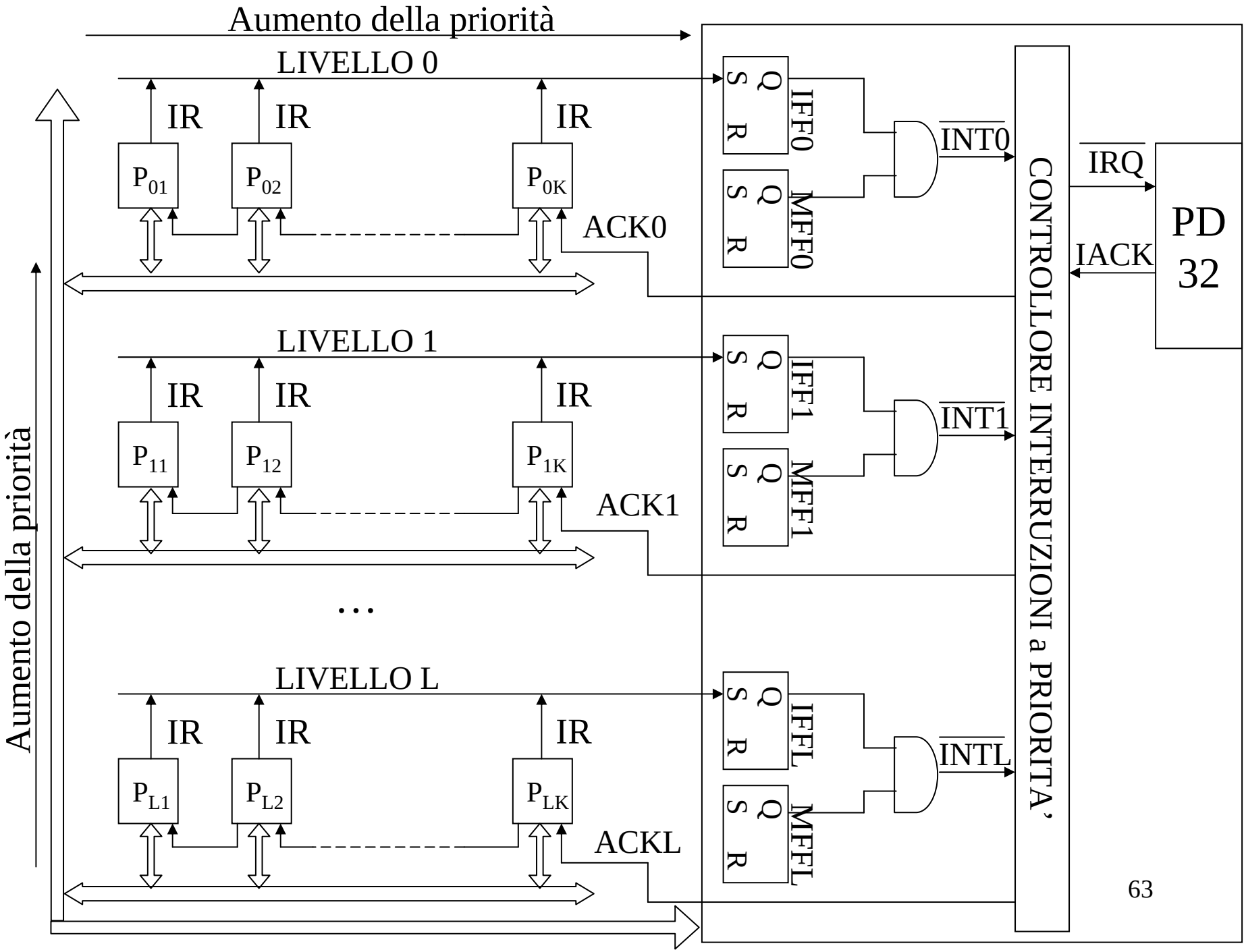
Driver: acquisizione di 100 dati tramite interruzione

```
INIT::
        MOVL #100,    AD1_C      ; valore iniziale del contatore
        MOVL #DATI,AD1_P         ;valore iniziale del puntatore
        START AD1                ;comando che inizia l'acquisizione del primo dato
        RET
;Routine di servizio
Driver 1, 600h
SAD1:   PUSH     R0              ;salva i registri usati
        PUSH     R1              ;
        MOVL AD1_C, R0           ; contatore
        MOVL AD1_P, R1           ; puntatore
        INW    AD1, (R1)+        ; trasferisce il dato in memoria e incrementa il
                                 ; puntatore
        SUBL     #1,  R0         ; decrementa il contatore
        JNZ       NEXT           ; era l'ultimo dato?
        CLEAR   AD1              ; si: rimuove la richiesta di interruz. rendendo il
        JMP       EXIT           ; dispositivo inattivo e ritorna al progr.interrotto
NEXT:   START  AD1               ; no: avvia l'acquisizione successiva
        MOVL    R0, AD1_C        ; aggiorna il contatore e…
        MOVL    R1, AD1_P        ; … il puntatore in memoria
EXIT:   POP   R1                 ; ripristina i registri usati
        POP   R0                 ;
        RTI                      ; ritorna al programma interrotto
```

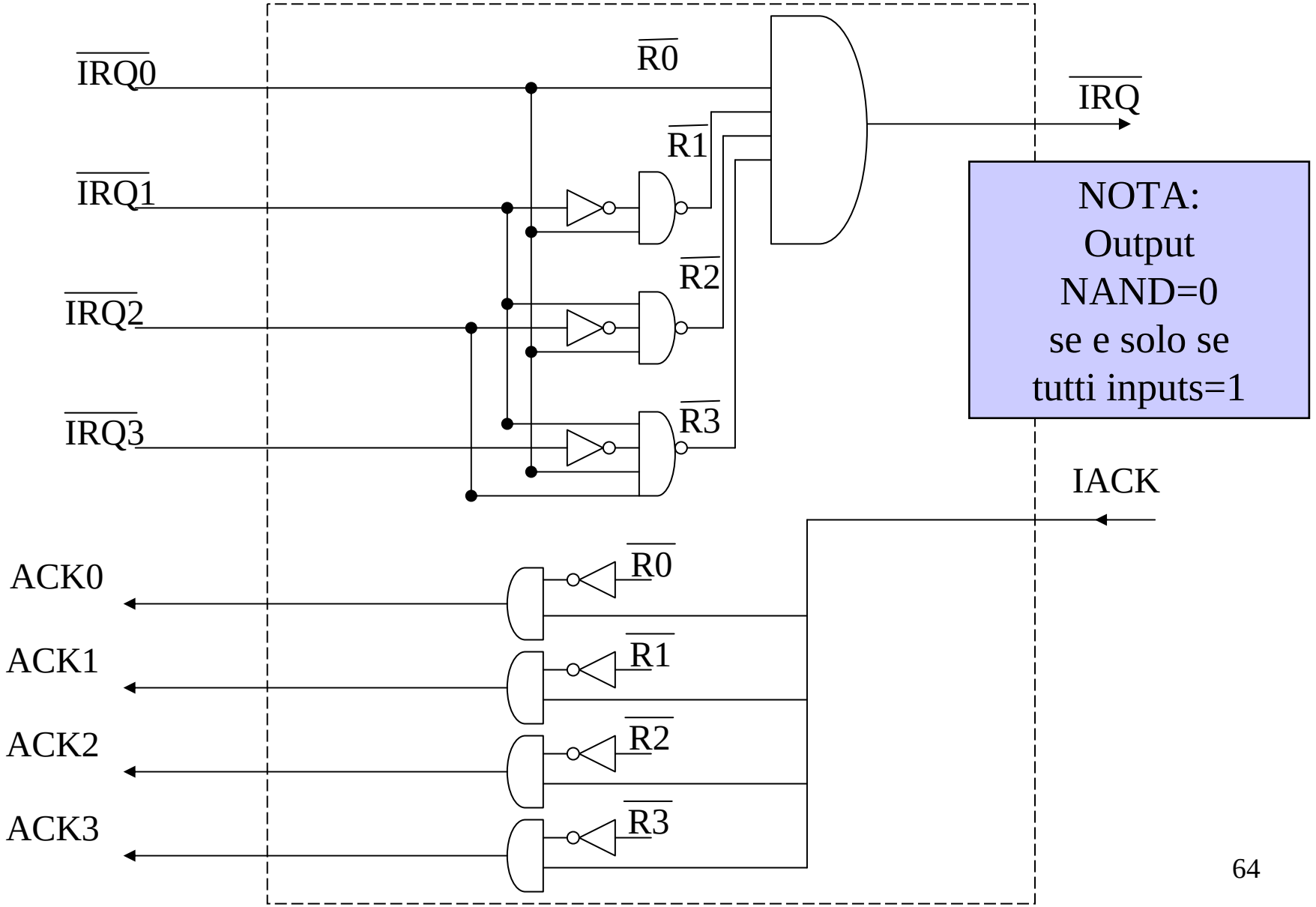
Concetto di azione atomica

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

messo a 1 dal microprogramma relativo a RTI


Aumento della priorità
LIVELLO 0
IR
IR
IR
P01
P02
P0K
ACK0
LIVELLO 1
P11
P12
P1K
ACK1
LIVELLO L
PL1
PL2
PLK
ACKL
Aumento della priorità
S
R
Q
IFF0
MFF0
IFF1
MFF1
IFFL
MFFL
INT0
INT1
INTL
CONTROLLORE INTERRUZIONI a PRIORITA'
IRQ
IACK
PD
32

Controllore interruzione a priorità per PD32



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 il

numero di istruzioni e quindi il 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

Operazioni di I/O gestite da canale

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

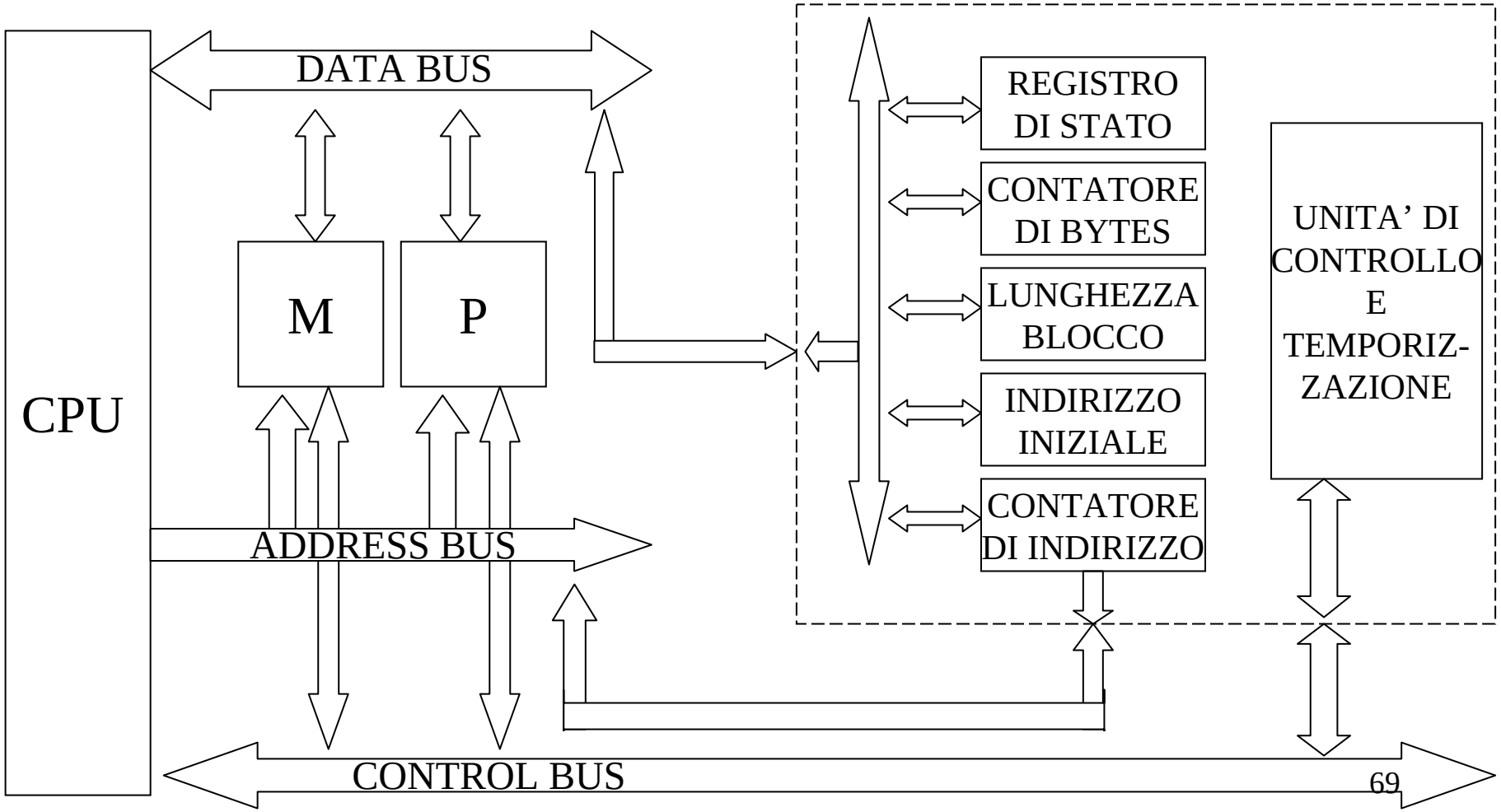
Una volta che il DMAC è stato programmato il processore lo deve attivare (p.e. tramite una START)

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 (CPU con un unico bus sia per la memoria che per I/O)



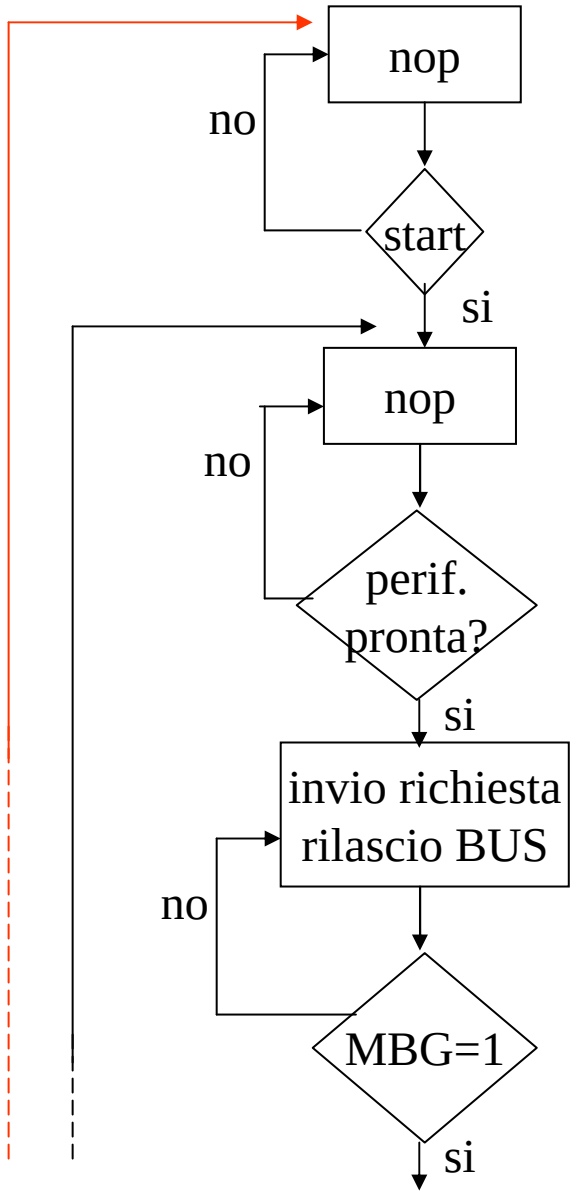
Protocollo di interazione DMAC-CPU

Trasferimento a Bus - stealing

DMAC

CPU

(1/2)



I/O

I/O

START

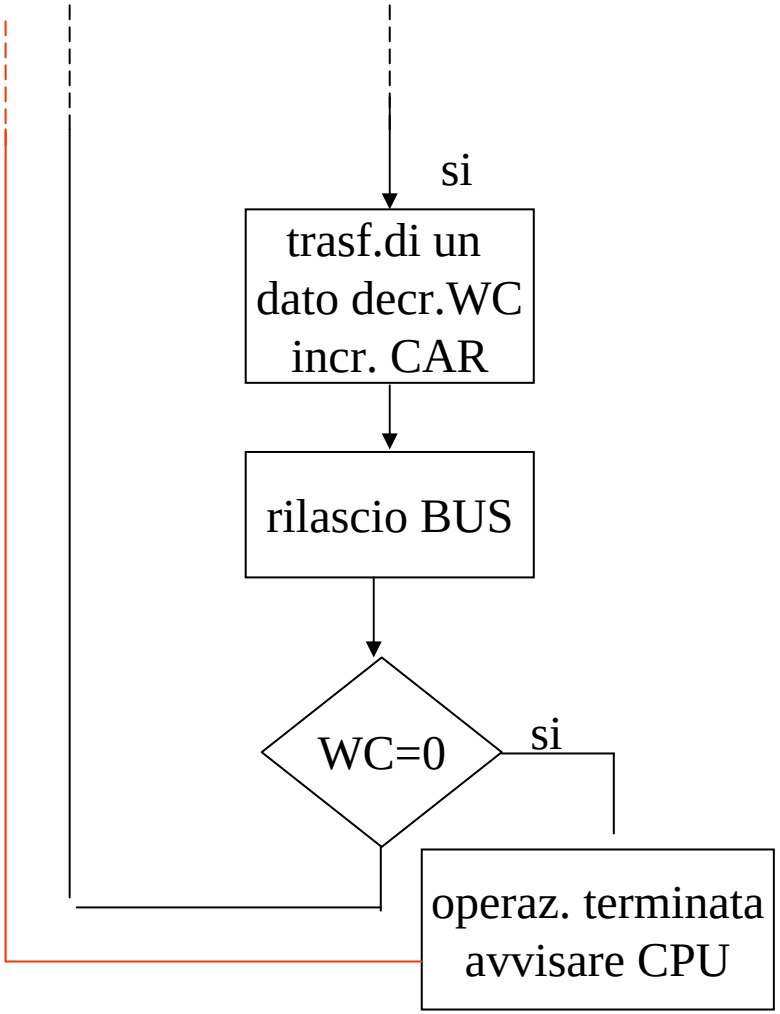
MBR

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

(2/2)



MBR

IRQ

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

La CPU esegue la routine di servizio

DMAC

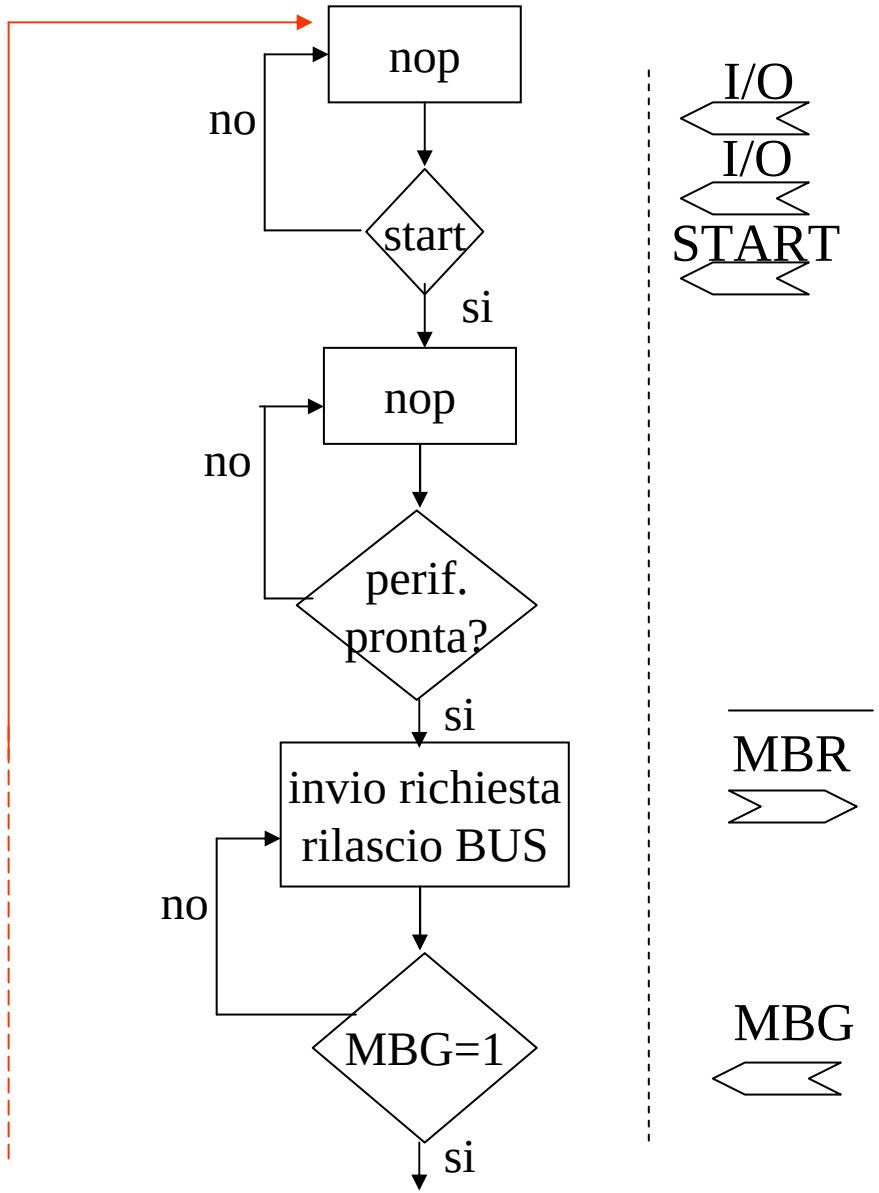
CPU

Protocollo di interazione DMAC-CPU

Trasferimento a BURST

DMAC

CPU



I/O

I/O

START

MBR

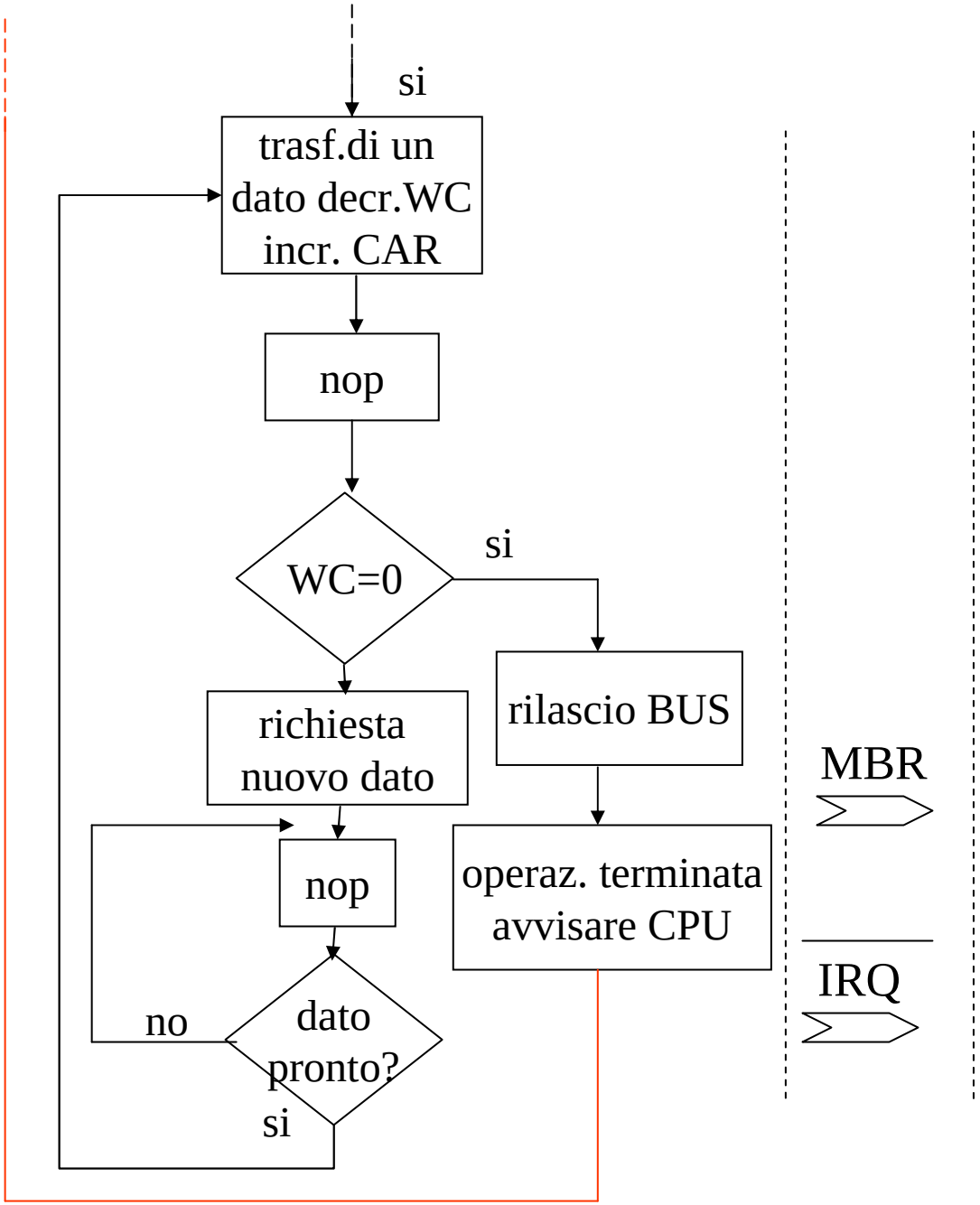
MBG

La CPU
Inizializza
Il DMAC

(1/2)

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

(2/2)

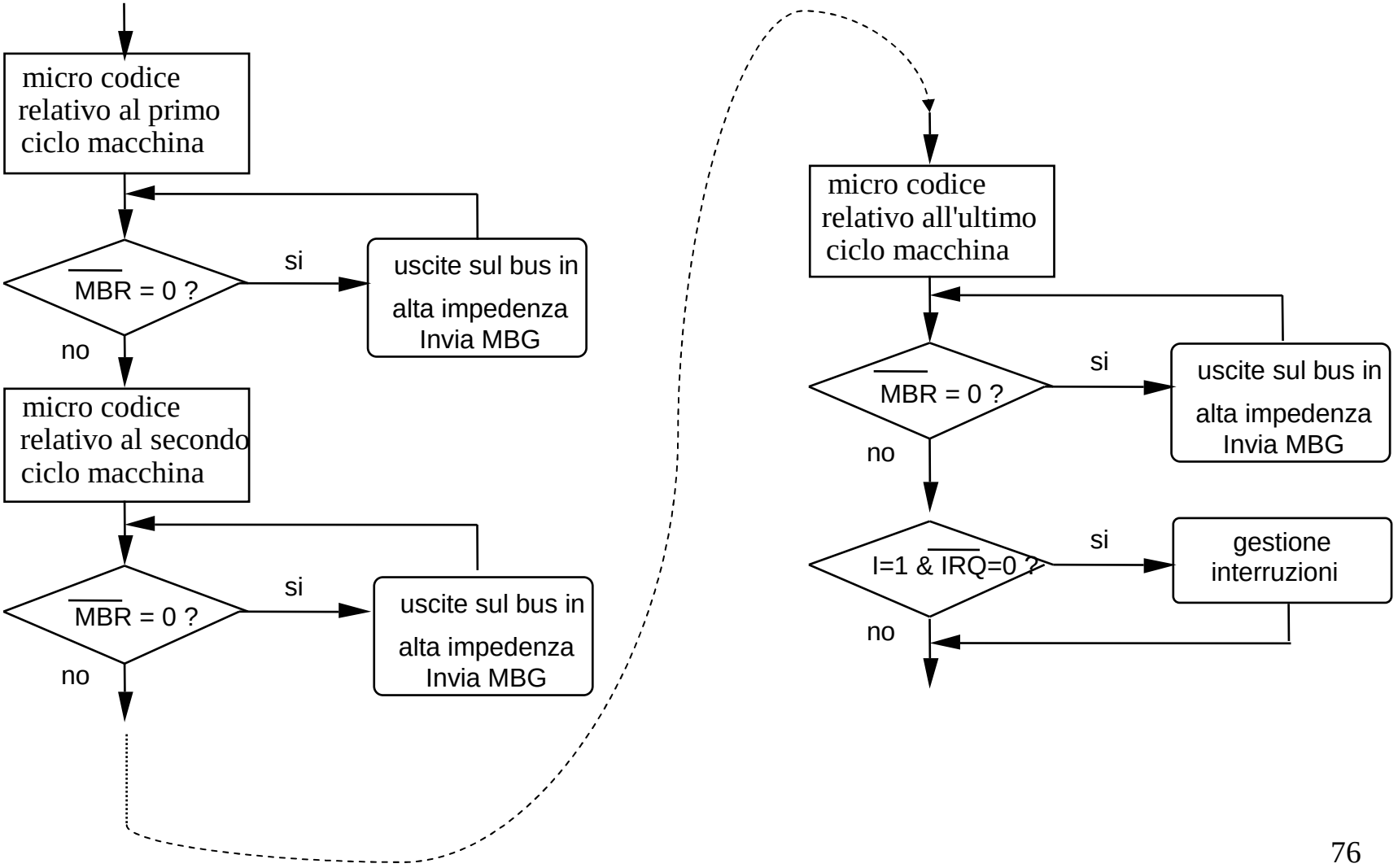


La CPU riprende il normale funzionamento *Riacquisto del controllo del BUS*

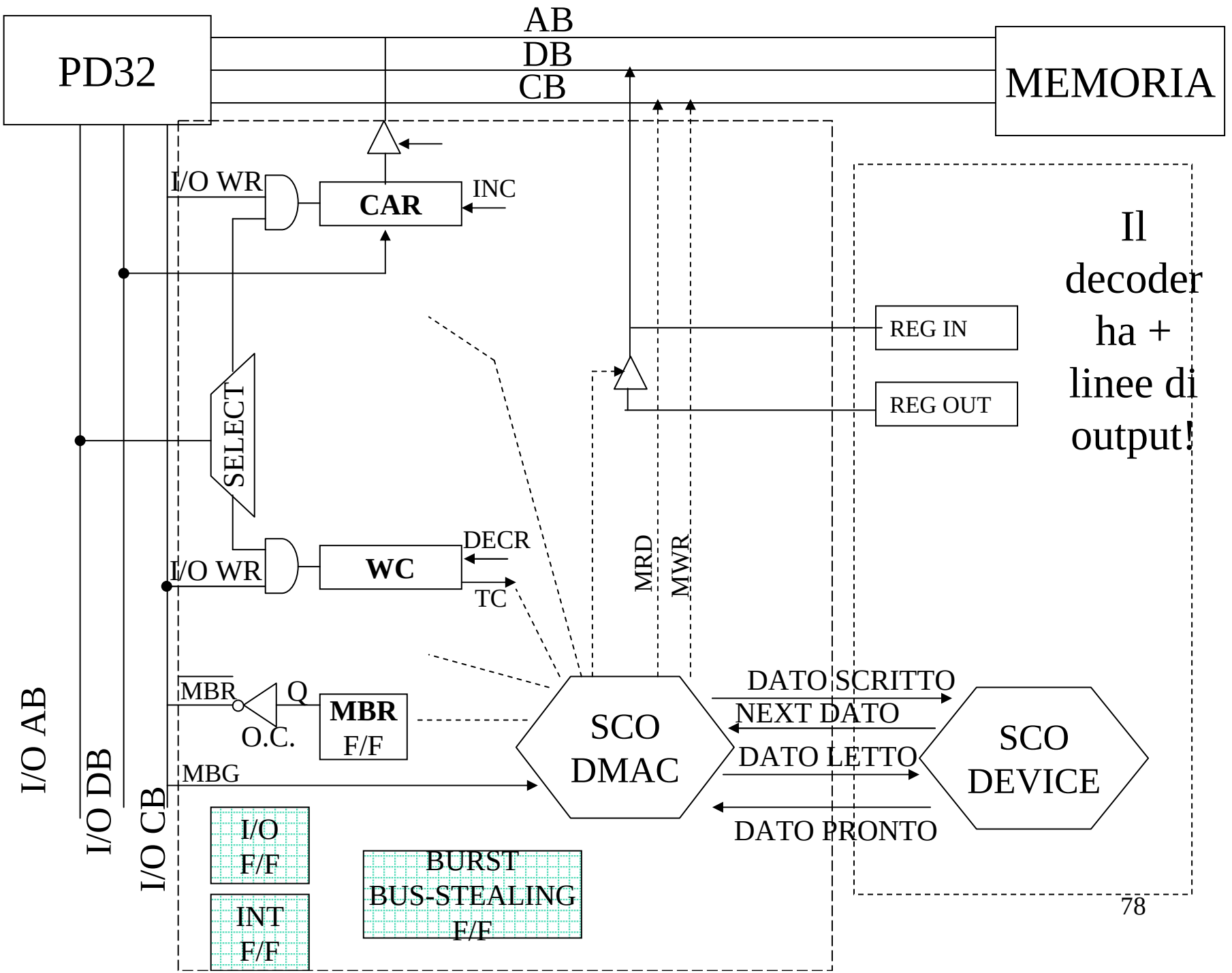
La CPU esegue la routine di servizio

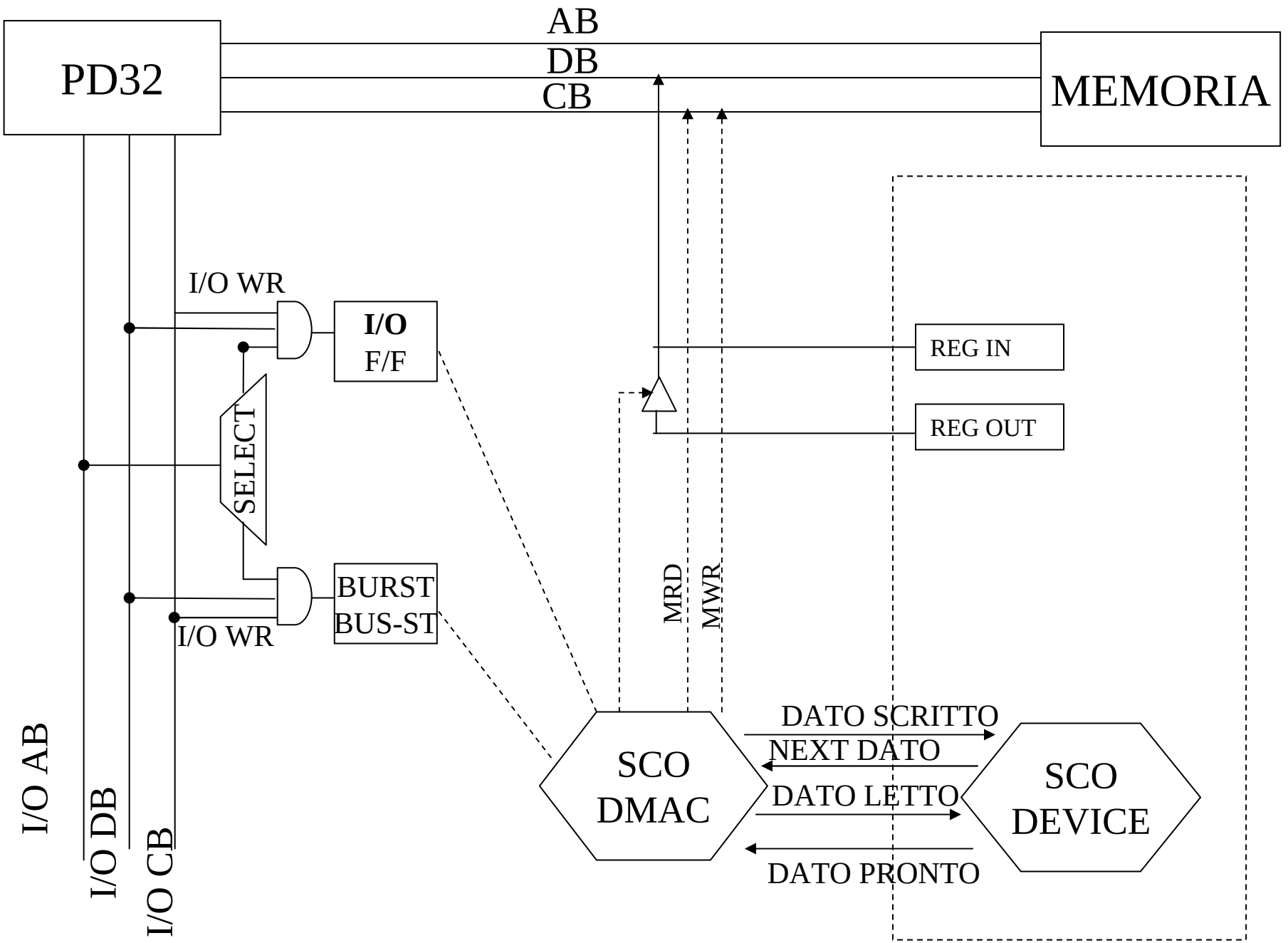
CPU

Modifica SCO PD32
per poter gestire le due modalità di interazione

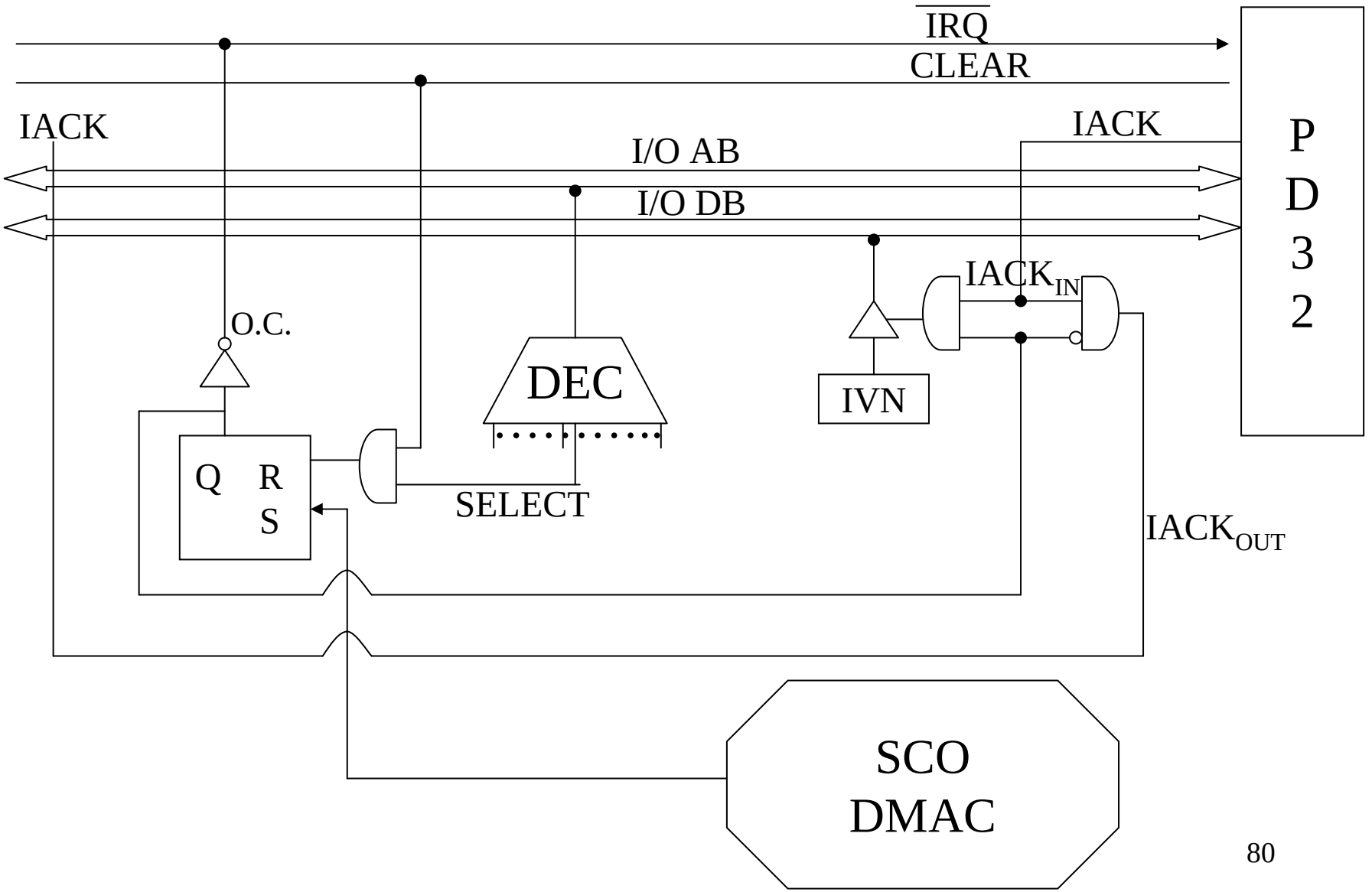


Architettura di un DMAC (adatto x PD32)
trasferimento dati tra una periferica e la memoria


PD32
AB
DB
CB
MEMORIA
I/O WR
CAR
INC
Il decoder ha + linee di output!
REG IN
REG OUT
SELECT
DECR
I/O WR
WC
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


PD32
AB
DB
CB
MEMORIA
I/O WR
I/O
F/F
SELECT
REG IN
REG OUT
BURST
BUS-ST
I/O WR
MRD
MWR
I/O AB
I/O DB
I/O CB
SCO
DMAC
DATO SCRITTO
NEXT DATO
DATO LETTO
DATO PRONTO
SCO
DEVICE

Il Segnale di START è omesso per semplicità



Esempio inizializzazione DMAC

```
INIZIALIZ:   MOVL #100,R0          ;carica in R0 il numero di word da leggere
             OUTLR0, WCOUNTER      ;e passa il valore al WC (nel DMAC)
             MOVL #2000,R0         ;carica in R0 l'indirizzo da cui leggere
             OUTL R0, CAREGISTER   ;e passa il valore al CAR (nel DMAC)
             MOVL #1h,R0
             OUTL R0, DMACI/O      ;programma il DMAC per la lettura
             MOVL #0,R0
             OUTL R0, DMACB-ST     ;seleziona la modalità (bus-stealing/burst)
             START DMAC            ;avvia trasferimento
```