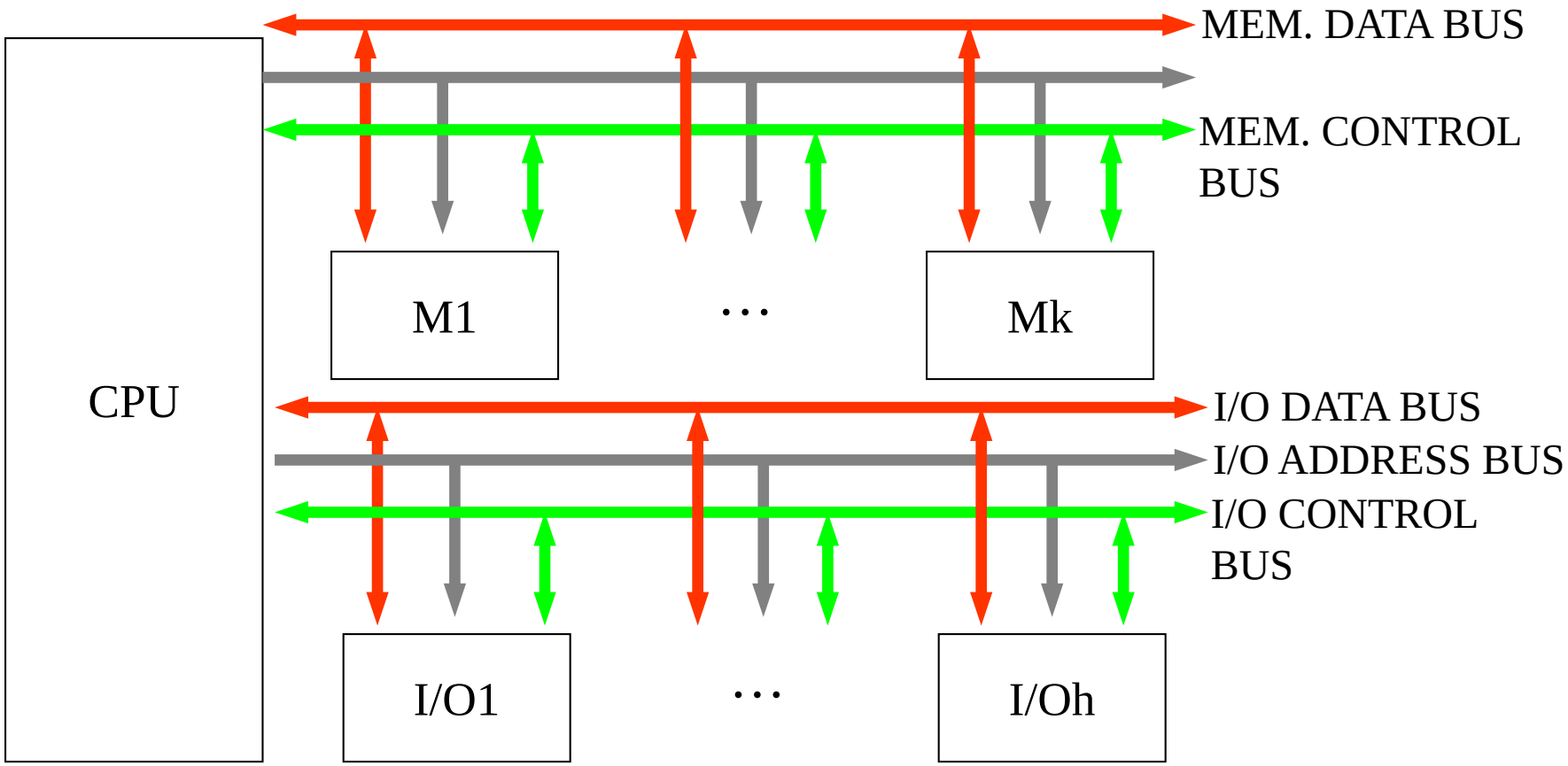
Processore con un unico bus

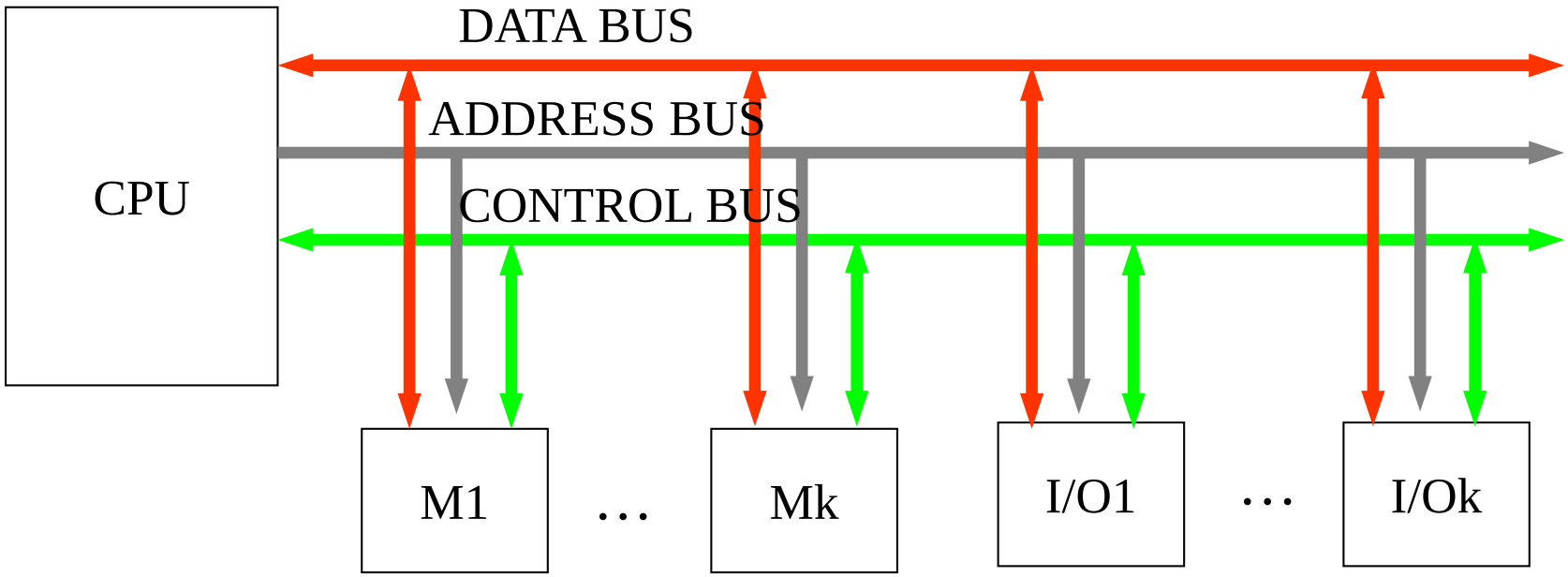
I soluzione con due bus distinti

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



II soluzione con solo un bus

Architettura ad un solo bus



Memory-mapped I/O

- Lo spazio di indirizzamento dell'I/O appartiene allo stesso spazio di indirizzamento della memoria
  - I registri, compresi i flip/flop, dei vari device controller sono considerati logicamente come locazioni di memoria, pur essendo fisicamente localizzati all'interno del device controller

Istruzioni dedicate

- Lo spazio di indirizzamento di I/O è separato dallo spazio di indirizzamento della memoria
- Per consentire al processore di accedere ai registri dei controller delle periferiche vengono inserite delle *istruzioni specifiche* nell'insieme delle istruzioni, dedicate alla gestione dell'I/O
- Queste istruzioni dedicate fanno riferimento esplicitamente al dispositivo interessato all'operazione di I/O
- Sul control bus oltre al segnale ***RD*** e ***WR*** è previsto un altro segnale di controllo: ***I/O***
- Se ***I/O*** = 1 si interagisce con una periferica altrimenti con la memoria

I device controller devono essere quindi dotati di componente che permetta loro di riconoscere le transazioni ad essi indirizzate

I controller sono progettati in modo che diventino parte attiva (in ingresso o in uscita) solo quando identificano sul bus uno degli indirizzi corrispondenti alle proprie porte (*bus snooping*)

Come visto nel caso di presenza di due bus il riconoscimento dell'indirizzamento può essere effettuato tramite un decoder

NOTARE CHE IL BUSY WAITING NON LO IMPLEMENTEREMO A FIRMWARE, perché BLOCCANTE A LIVELLO SOFTWARE E, QUINDI, NON POTREBBE CONSENTIRE DI UTILIZZARE LA RISORSA PROCESSORE NEL CASO DI INTERAZIONE CON PERIFERICHE LENTE, NON A CASO TALE MODALITà NON è PREVISTA NEANCHE NELL'X86 E CIO' CONSENTE DI POTER «UTILIZZARE» LE FUNZIONALITà DEL SISTEMA OPERATIVO.

Si continuano ad usare i segnali di controllo: RD, WR e I/O

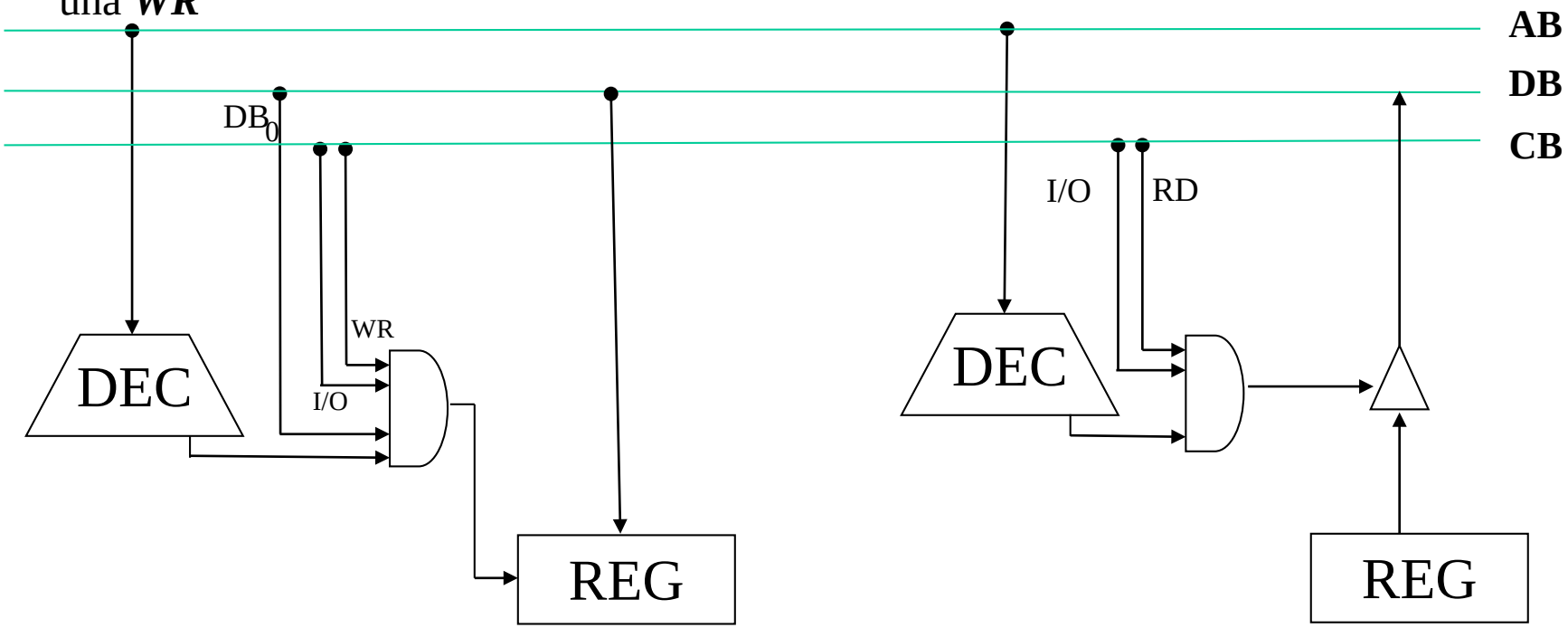
- RD x Read da memoria o da dispositivo
- WR x Write su memoria o su dispositivo
- I/O per selezionare tra memoria e dispositivo esterno

Da notare che prima I/O era solo sul I/OCB ora viene usato anche per selezionare la memoria (quando I/O=0), inoltre non si usano più i segnali di controllo MRD e MWR che si usavano nel caso di due bus

Del vecchio I/O control bus rimangono ancora i segnali di INT, INTA, HOLD e HOLDA che saranno integrati nell'unico control bus previsto sia per la memoria che per i dispositivi esterni

Come già detto la presenza di numerosi dispositivi connessi allo stesso bus comporta l'allungamento delle dimensioni dei fili del bus stesso, questo comporta, se si desidera utilizzare interagire in modalità sincrona, di utilizzare clock a bassa frequenza, rallentando l'interazione tra il processore e la memoria di lavoro, anche se i dispositivi di ingresso uscita potrebbero utilizzare protocolli asincroni. Naturalmente questa soluzione, seppure fattibile, non viene normalmente utilizzata per connettere grandi quantità di dispositivi, qui viene presentata solo **a fini didattici**, dopodiché una volta capiti i fondamenti delle interazioni con questa architettura, si farà vedere come, sempre utilizzando un unico bus che connette il processore con il mondo esterno, sia possibile diminuire le dimensioni fisiche dei bus per consentire di incrementare la frequenza di funzionamento del bus (ciò potrà essere realizzato nel caso migliore mettendo **un unico dispositivo** oltre alla memoria di lavoro)

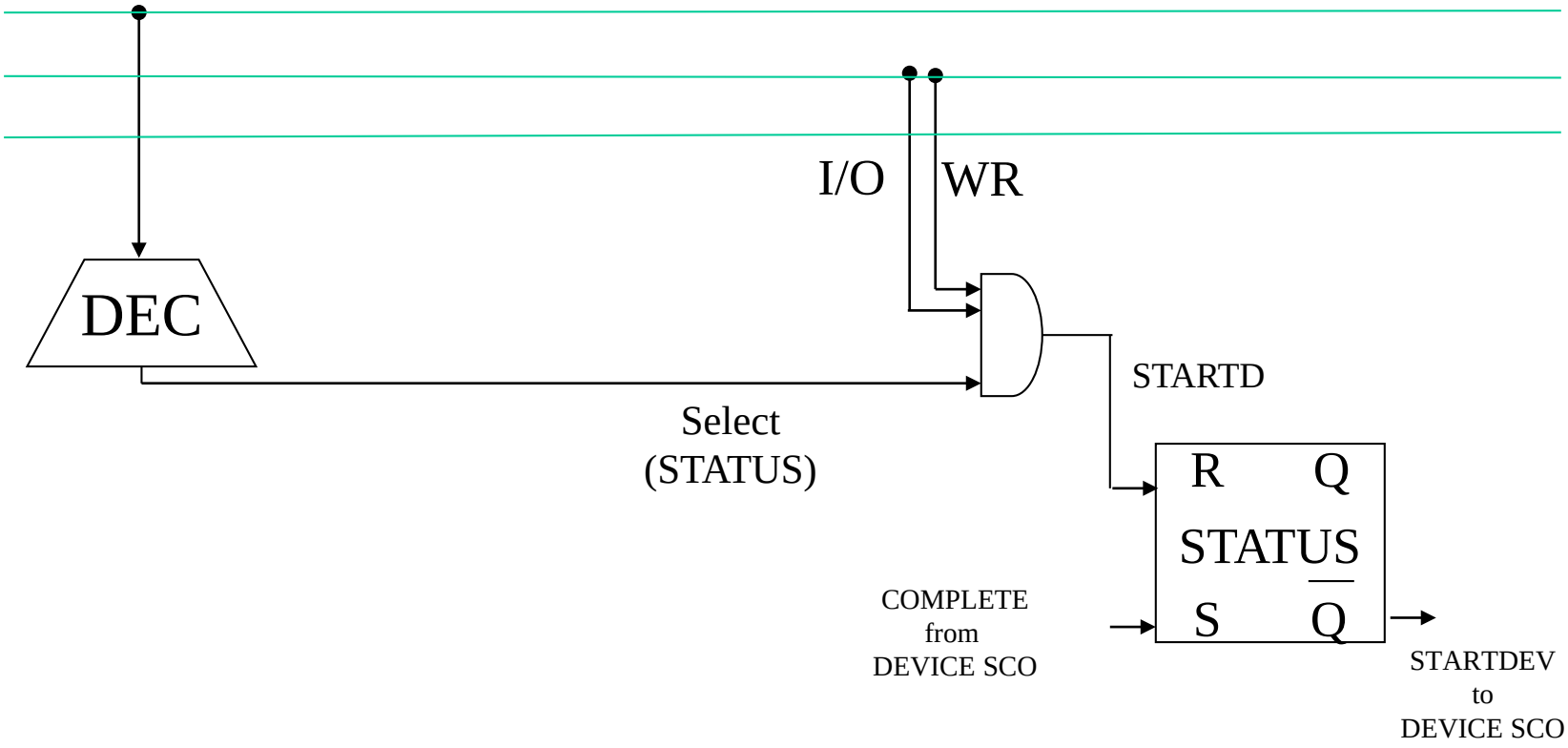
- Le istruzioni ***IN*** e ***OUT*** vengono implementate a firmware in modo similare a quanto fatto nel caso di presenza di due bus, in questo caso dato che si usa lo stesso address bus sia per indirizzare la memoria che i dispositivi di ingresso/uscita si dovrà introdurre un segnale di controllo che dirà se le informazioni che si trovano sull'address bus sono relative alla memoria o ai dispositivi di ingresso/uscita, tale segnale di controllo di seguito lo indicheremo come I/O, quindi se ***I/O*** = 1 vuol dire che l'indirizzo sull'address bus è relativo ad un dispositivo esterno, se ***I/O*** = 0 l'indirizzo è relativo alla memoria,
- Naturalmente per leggere il microprogramma genererà una ***RD***, mentre per scrivere una ***WR***



Per interagire con i flip/flop della periferica, ed in particolare con INT_REQ e STATUS si utilizzano delle istruzioni di ***out***, mentre per leggere l'uscita del flip/flop STATUS si utilizza una ***in***, come di seguito indicato e già fatto nel caso di processore con due bus.

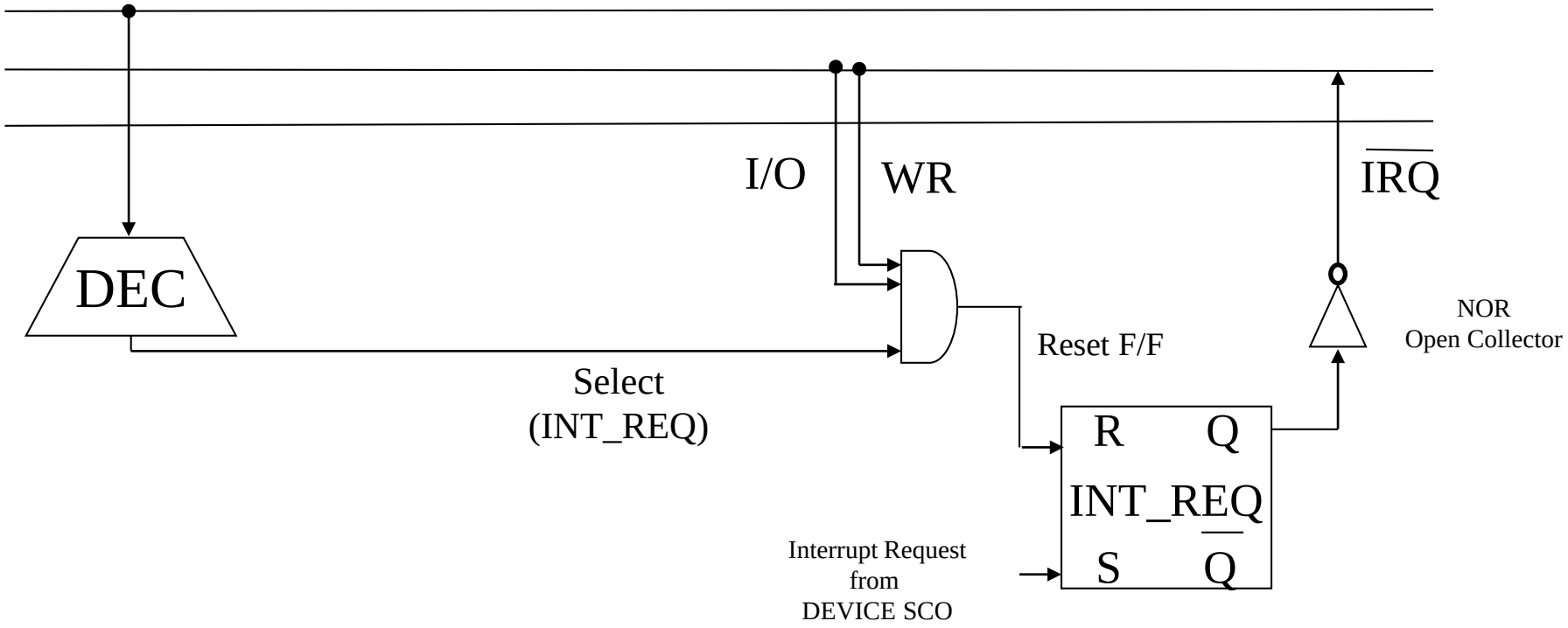
Per resettare flip/flop STATUS

MOVW $FF_STATUS, %dx
MOVB $1, %al
OUTB %al, %dx



Per resettare flip/flop INT_REQ

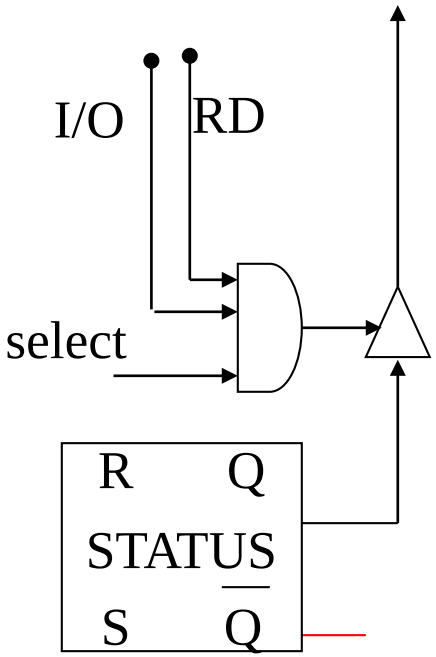
MOVW $INT_REQ, %dx
MOVB $1, %al
OUTB %al, %dx



- Per verificare che una periferica è pronta si legge l'uscita del flip/flop STATUS e si verifica se è pari ad 1 o meno

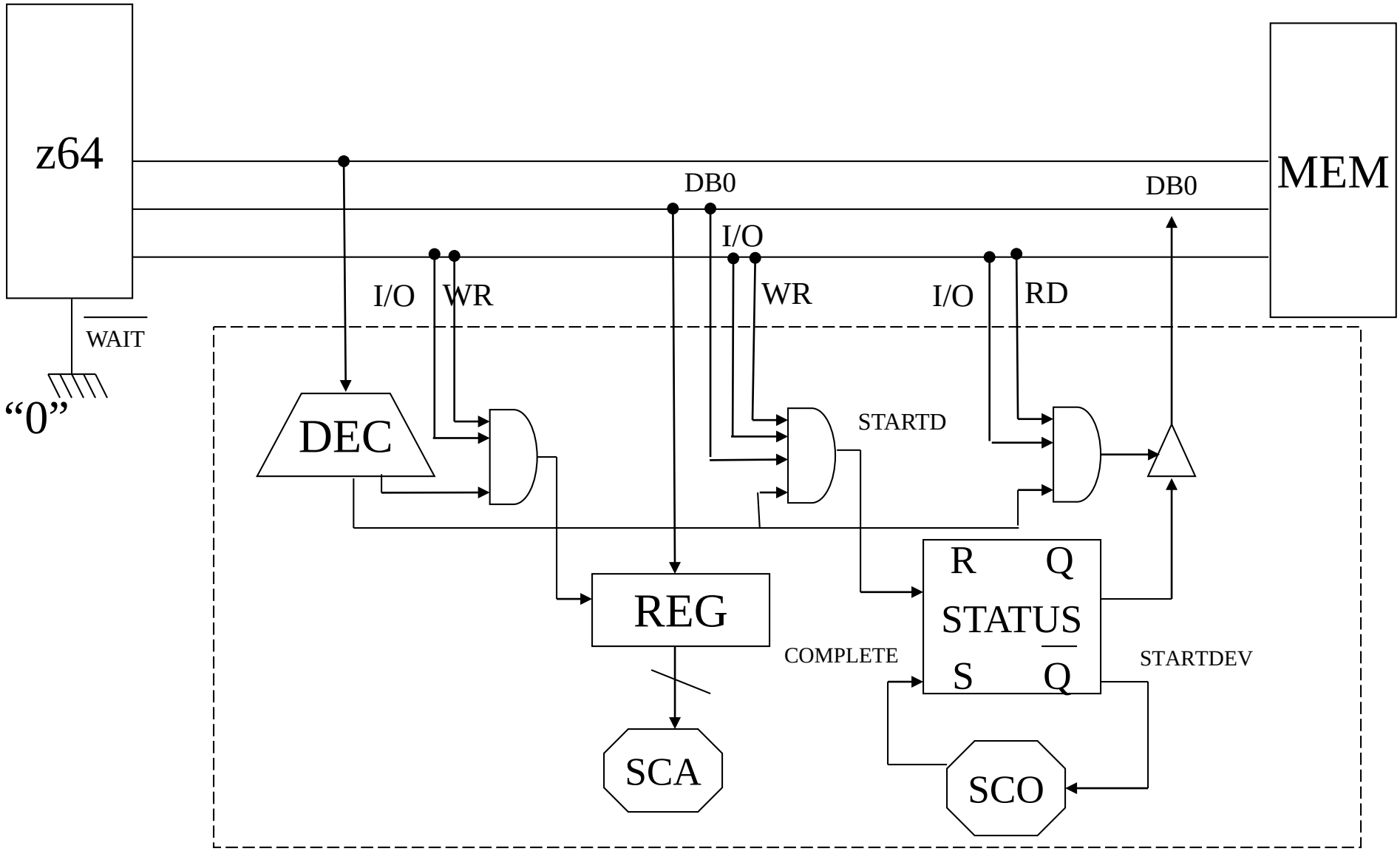
- Esempio di attesa fino a che il carry non è pari ad uno

```
MOVW $STATUS, %dx
Aspetta: INB %dx, %al
         BTB $0, %al # copia il bit n.0 del reg.
                     # A nel carry
         JNC Aspetta
```



I/O programmato- INTERFACCIA di OUTPUT

PROTOCOLLO DI HANDSHAKING IMPLEMENTATO A SOFTWARE



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, il valore 1 nella linea 0 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 carry (JC).
4. Se CARRY= 0 il processore deve attendere ed eventualmente tornare al punto 3.
5. Se READY= 1 il processore può eseguire un'altra istruzione.

Porzione di programma assembly handshaking per un dato

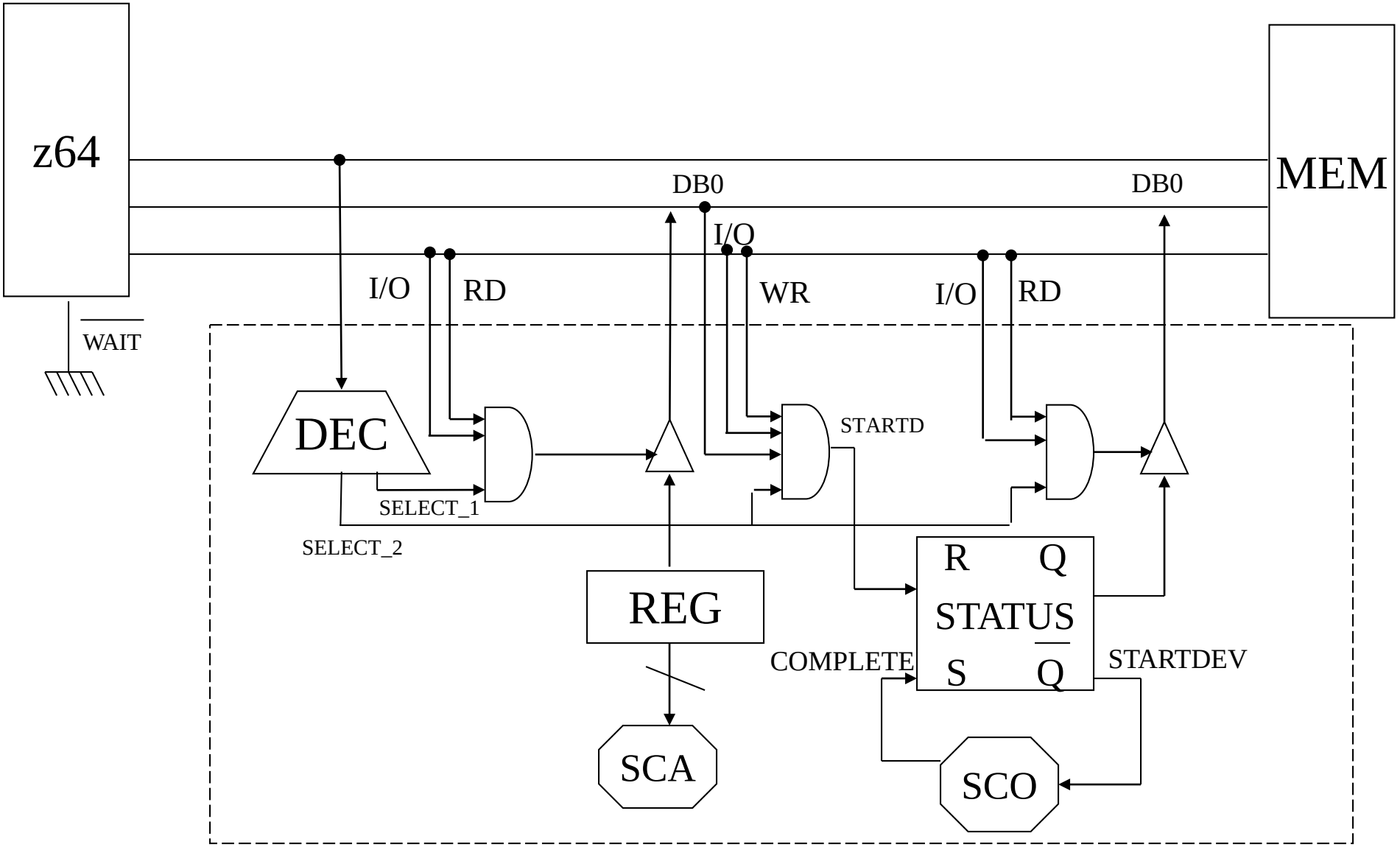
```
        MOVL $dato, %eax
        MOVW $DeviceOUT, %dx
        OUTL %eax, %dx                  # out

        MOVW $FF_STATUS, %dx
        MOVB $1, %al
        OUTB %al, %dx                   # start

Aspetta: INB %dx, %al
        BTB $0, %al             # copia il bit n.0 del reg.
                                # AL nel carry
        JNC Aspetta                     # jnr
```

I/O programmato – INTERFACCIA di INPUT

PROTOCOLLO DI HANDSHAKING IMPLEMENTATO A SOFTWARE



I/O programmato – INTERFACCIA di INPUT

PROTOCOLLO DI HANDSHAKING 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 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).

Porzione di programma assembly handshaking per un dato

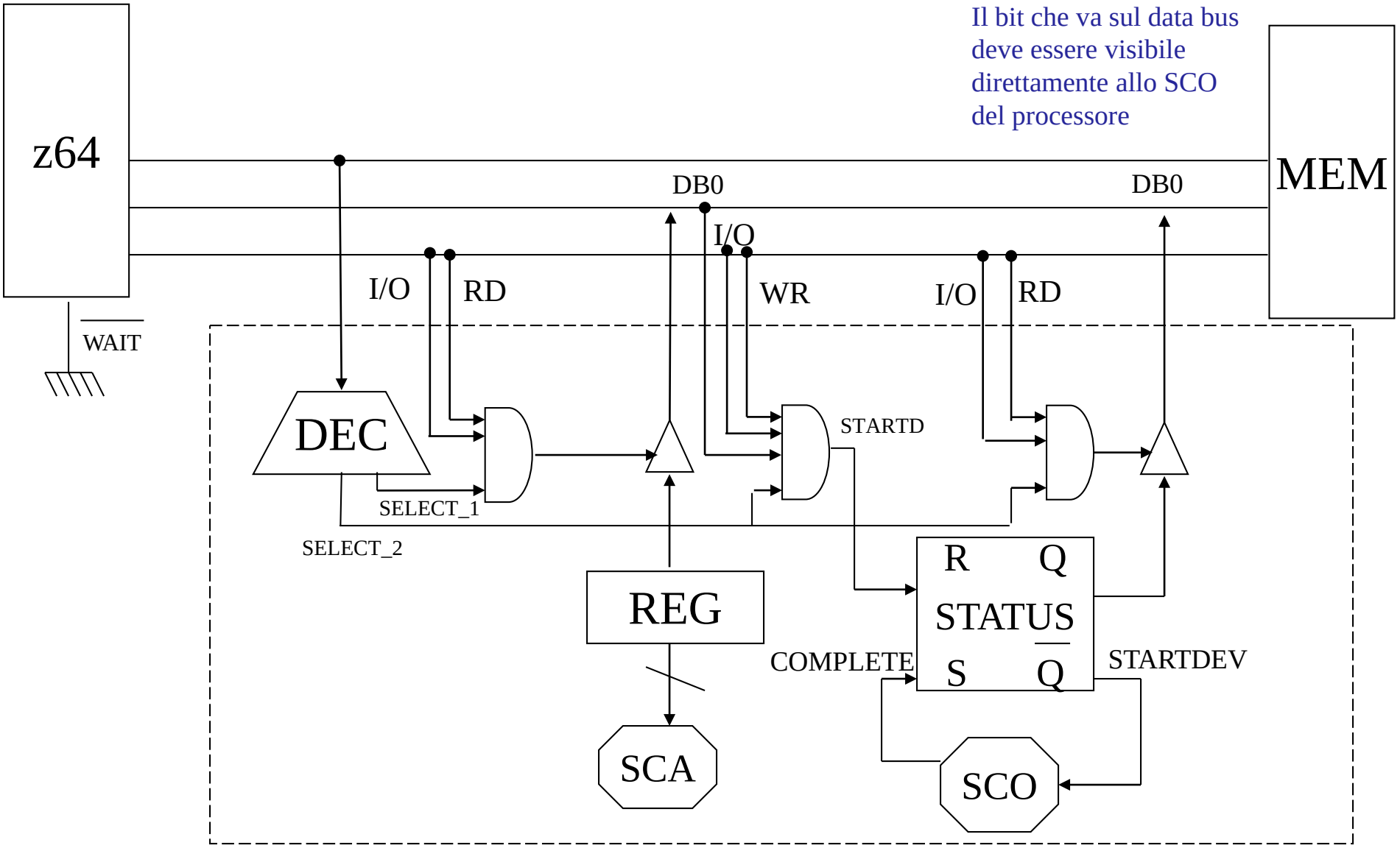
```
                MOVW $FF_STATUS, %dx
                MOVB $1, %al
                OUTB %al, %dx          # avverti la periferica
Aspetta:        INB %dx, %al
                BTB $0, %al            # copia il bit n.0 del reg.
                                        # A nel carry
                JNC Aspetta            # attendi per. sia pronta
                MOVW $DEVICE_IN, %dx
                INL %dx, %eax          # input da periferica
```

Implementazione di INSX e OUTX

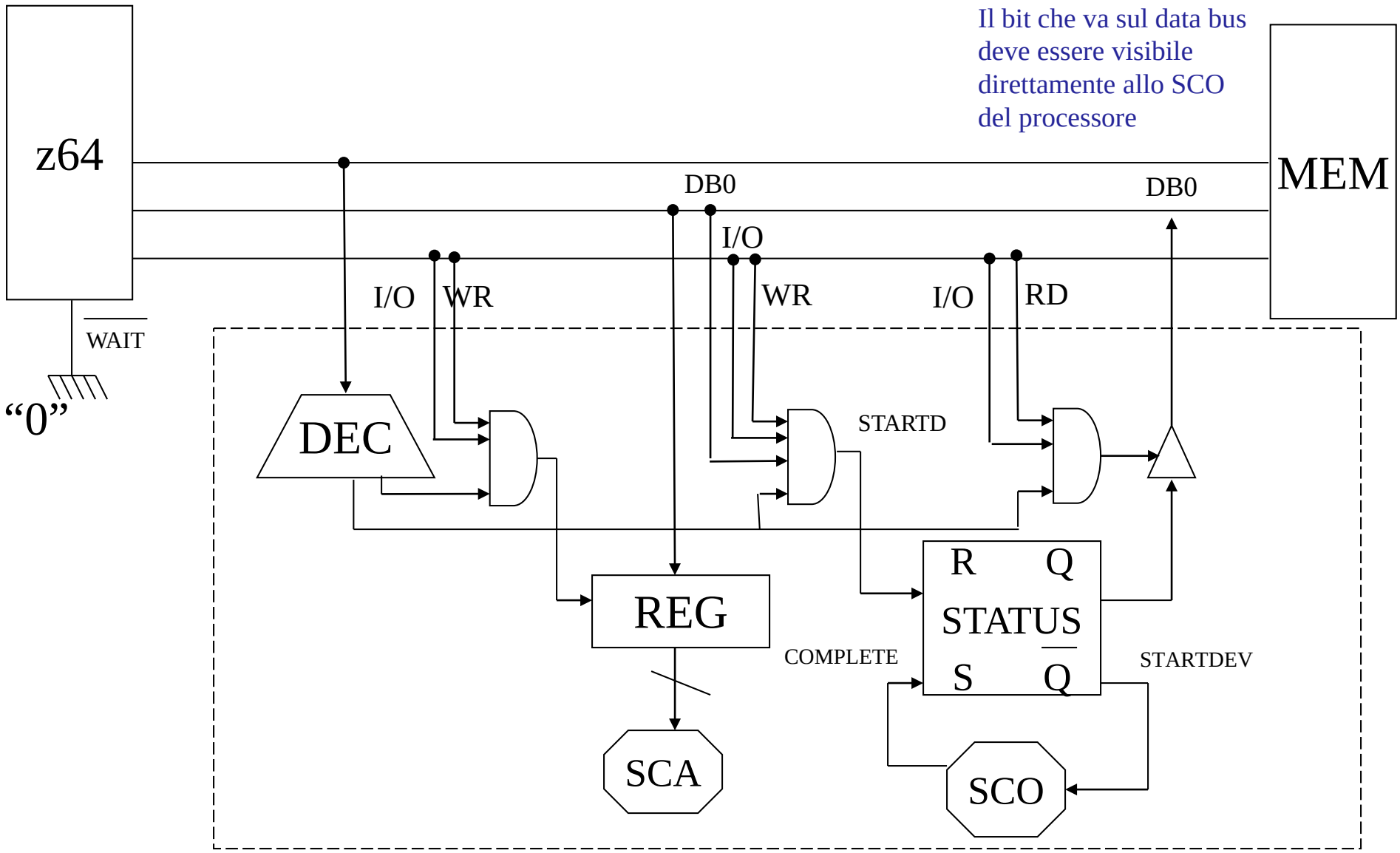
L’ INS e l’OUTS si possono implementare:

- a firmware emulando il trasferimento di N dati in modalità busy_waiting, come già fatto vedere nel capitolo precedente quando c’erano due bus distinti, con l’accortezza di non sporcare lo Status Register.
- utilizzando un DMAC in modalità burst (il DMAC già l’hanno visto con due bus), programmato con un microprogramma. Da notare che il segnale di hold viene testato ogni ciclo macchina e quindi ci sarebbe il rischio che una volta dato lo «start» al DMAC questo non mandi in tempo l’hold request e il processore eseguirebbe altre istruzioni prima di essere messo in uno stato di hold, per questo motivo dopo aver dato avviata la periferica è opportuno mettere una istruzione di verifica del trasferimento, emulando una busy waiting, in tal caso non è necessario utilizzare l’interrupt, come nel caso con due bus.

Hw per interagire con una periferica con una insx (busy waiting)



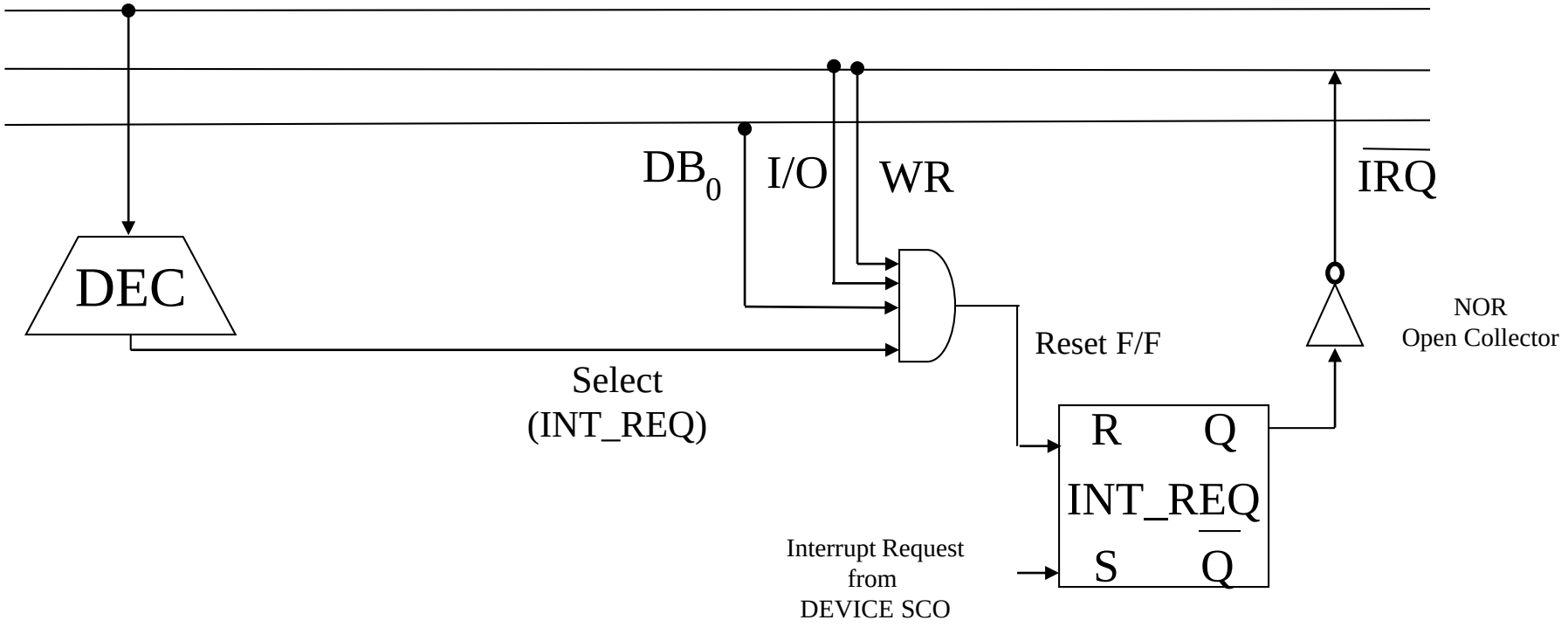
Hw per interagire con una periferica con una outsx (busy waiting)



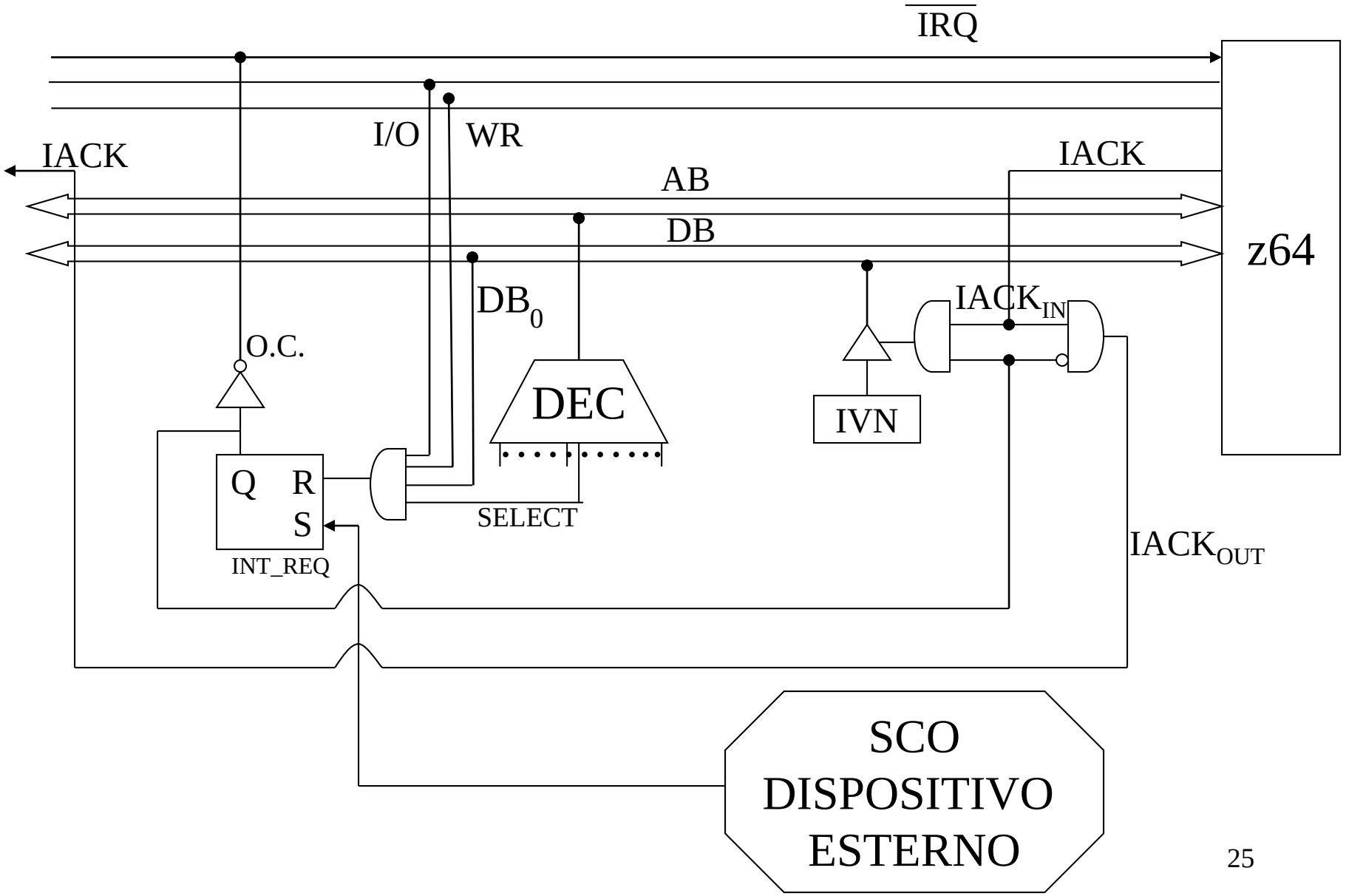
Gestione degli interrupt

- Viene fatta come nel caso di due bus

Richiesta interruzione e resettaggio F/F INT_REQ



Generazione interrupt, identificazione programma di servizio e resettaggio F/F INT_REQ



Nella presente implementazione del processore si è ipotizzato che il processore fosse in grado di interagire con un unico bus dedicato verso le periferiche oltre naturalmente che avesse la possibilità di interagire con la memoria.

Ciò comporta che:

- le connessioni fisiche del bus sono lunghe e quindi la velocità di trasferimento è limitata
- tutte le periferiche devono presentare la stessa interfaccia verso il processore, questo implica che ogni costruttore di periferiche deve conoscere il protocollo di comunicazione del processore per potersi interfacciare, protocollo che normalmente è proprietario.

Per:

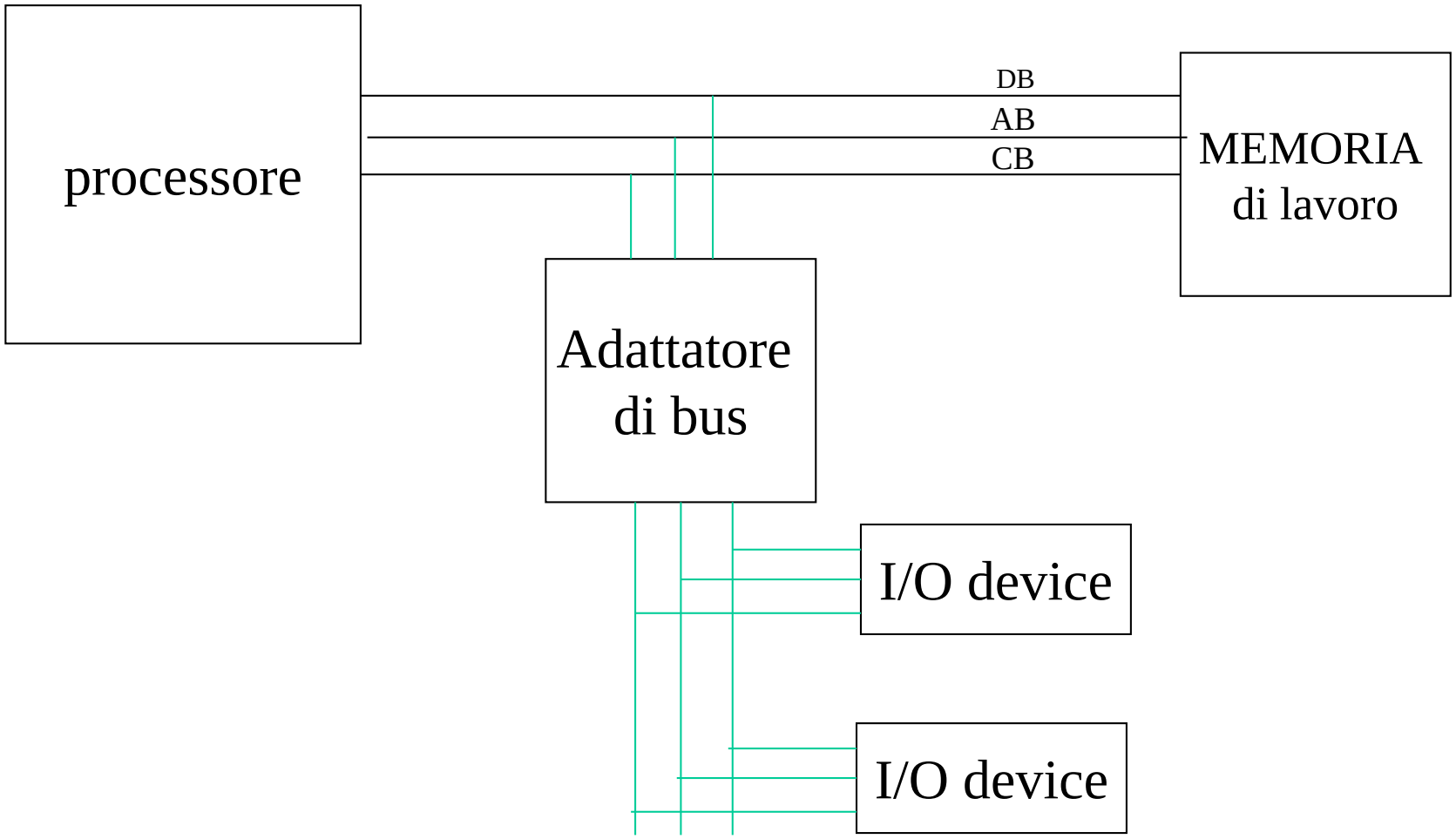
- Aumentare la banda di trasferimento dei dati da e verso la memoria

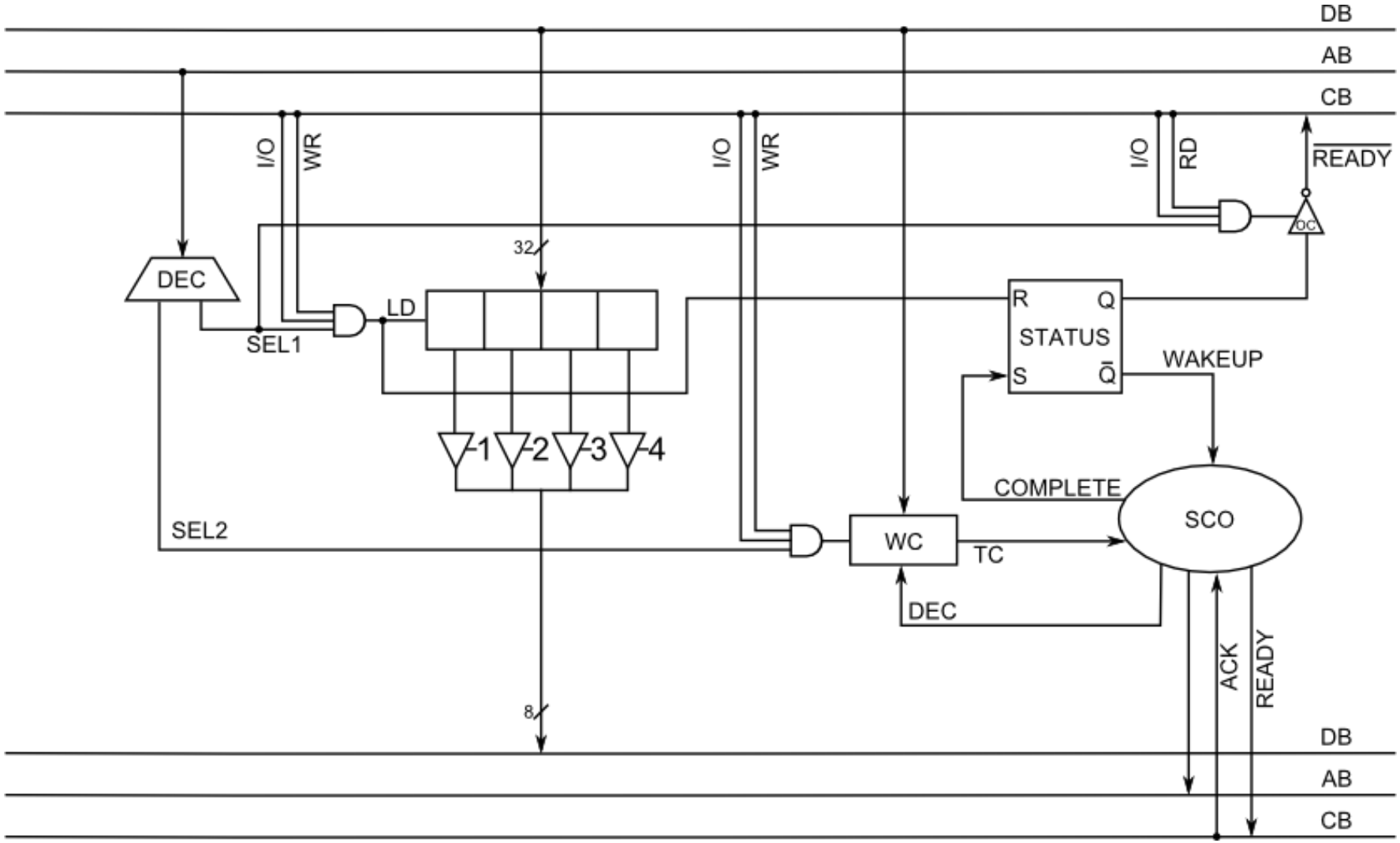
- Permettere ai costruttori di periferiche e ai progettisti dei processori di progettare le proprie interfacce nel modo più indipendente possibile, vedere i bus standard presentati precedentemente.

Si sono utilizzati processori di input/output dedicati che effettuano le operazioni di trasferimento dei dati dal processore di calcolo alle periferiche e viceversa, oppure dalla memoria di lavoro alle periferiche o viceversa

Di seguito faremo vedere soluzioni sempre più complesse partendo dalle più semplici per arrivare a soluzioni architetturali simili a quelle che si trovano utilizzate nelle architetture reali

Adattatore di bus
tra bus di memoria e verso bus di I/O del processore I versione)



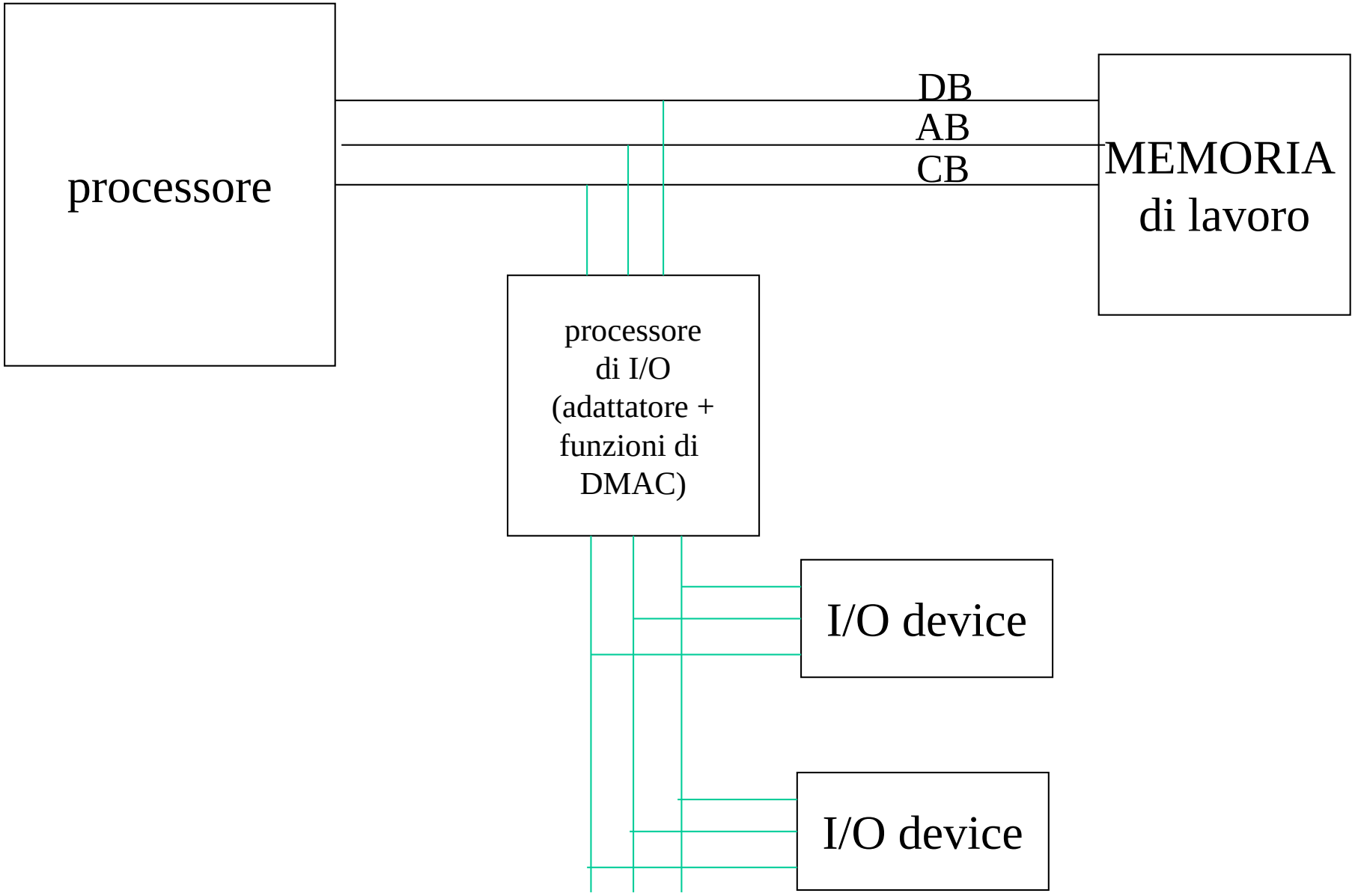

DB
AB
CB
I/O
WR
I/O
WR
I/O
RD
READY
OC
DEC
32
LD
SEL1
R
Q
STATUS
S
Q̄
WAKEUP
1
2
3
4
COMPLETE
SCO
SEL2
WC
TC
DEC
ACK
READY
8
DB
AB
CB

Nei primi 32 bit del messaggio da spedire vengono indicate il numero di “parole” contenute nello stesso, e questa quantità viene salvata nel WC. Nei secondi 32 bit viene indicato un numero identificativo di una periferica connessa al bus asincrono. Di questi bit vengono selezionati gli 8 meno significativi (abilitando il segnale “4“) e spediti al bus.

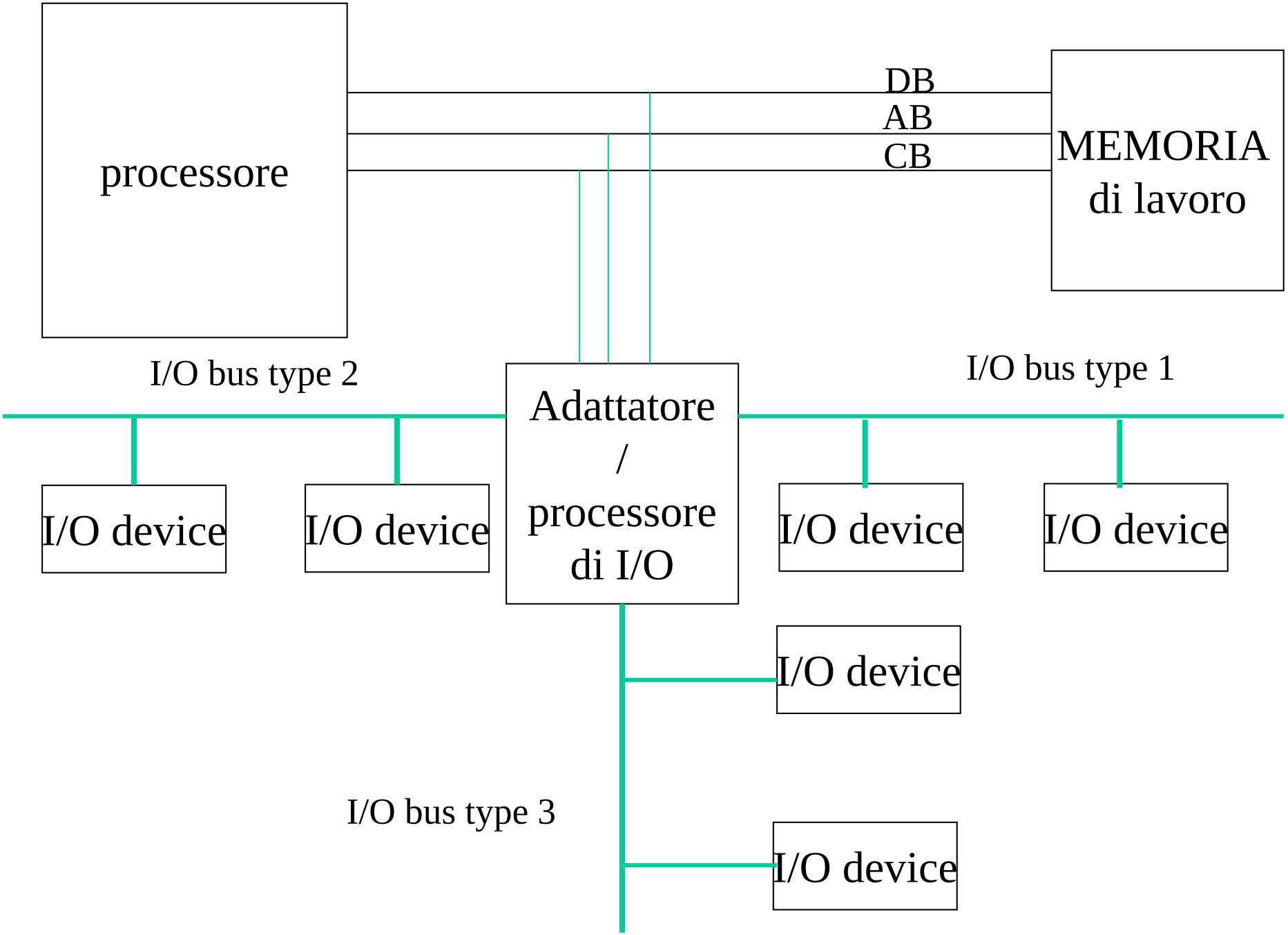
Da questo momento sino all'asserirsi di TC inizia il trasferimento controllato da un F/F che funge da semaforo, in quanto viene settato ad 1 non appena viene scritto un dato sul registro d'interfaccia, e ciò genera i segnali di WAIT e WAKEUP i quali rispettivamente mantengono in attesa il processore e “svegliano” lo SCO dell'adattatore. Appena viene consumato il dato lo SCO resetta il F/F, decrementa il WC e si mette in attesa di un nuovo dato dalla CPU. I 4 segnali di controllo “1”, ”2”, ”3” e ”4” e i relativi buffer three state fungono da multiplexer e selezionano un blocco di 8 bit dal registro di interfaccia (32 bit) per poter inviare il dato sul bus asincrono (8 bit). Sono necessari 2 segnali di selezione SEL1 e SEL2 in quanto ne occorre uno per il registro da 32 bit ed uno per il contatore.

- Sco dell’adattatore, vedere pdf

Processore di I/O con funzioni di DMAC


processore
DB
AB
CB
MEMORIA
di lavoro
processore
di I/O
(adattatore +
funzioni di
DMAC)
I/O device
I/O device

Versione più complicata: adattatore di più bus


processore
DB
AB
CB
MEMORIA
di lavoro
I/O bus type 2
I/O bus type 1
Adattatore
/
processore
di I/O
I/O device
I/O device
I/O device
I/O device
I/O device
I/O bus type 3
I/O device