PROGETTO

Due periferiche, *una di input ed una di output*, vogliono scambiare dati (in formato byte) tra di esse. Per supportare lo scambio, viene utilizzato un buffer (di dimensione 1 byte) in memoria di lavoro RAM.

La periferica PRODUTTORE (periferica di input) genera un dato che deve essere scritto all'interno del buffer tampone.

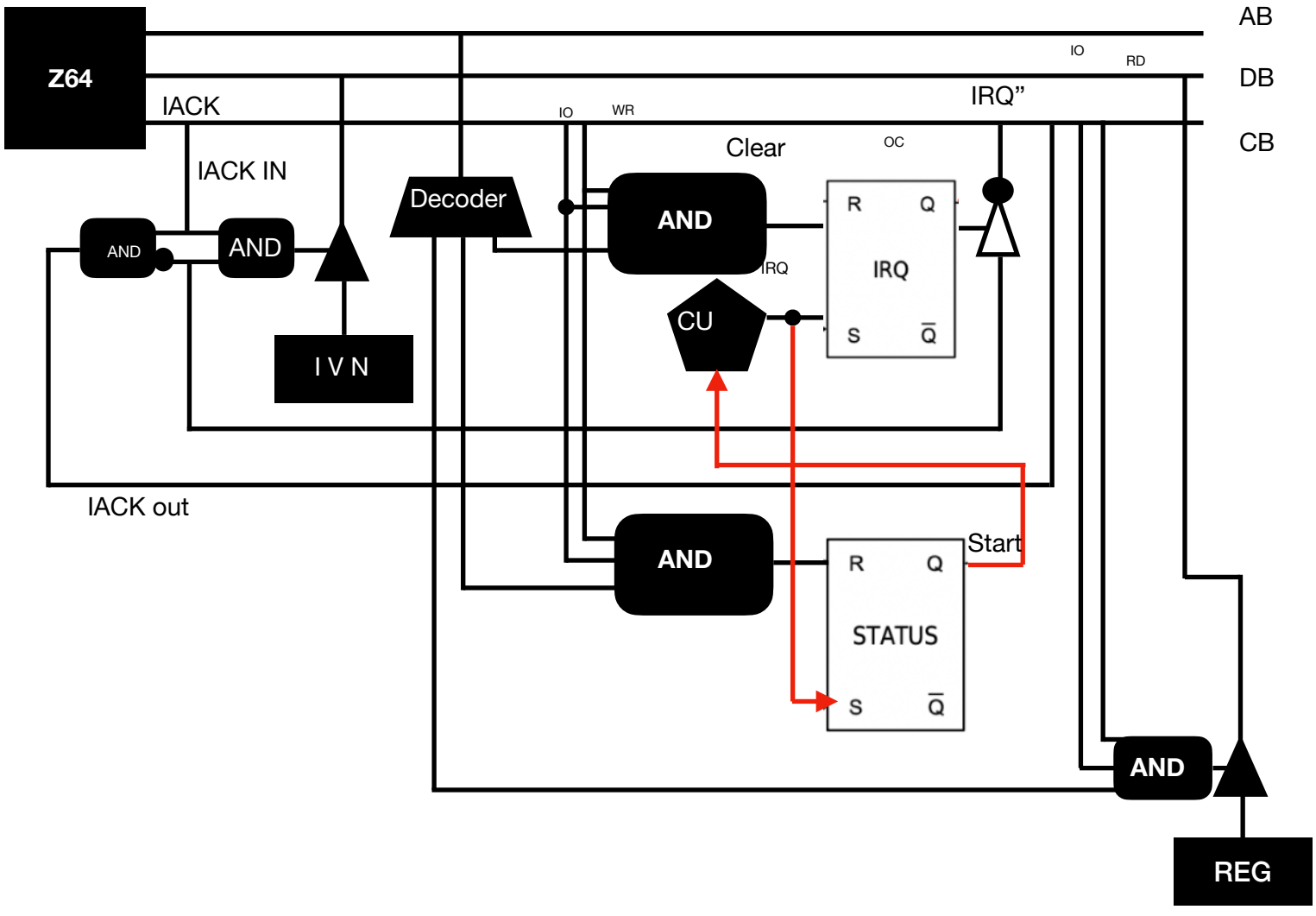
La periferica CONSUMATORE (di output) riceverà dal buffer il dato prodotto e lo processerà.

È necessario impedire alla periferica PRODUTTORE di generare un nuovo dato fintanto che quello contenuto all'interno del buffer tampone non sia stato correttamente processato dalla periferica CONSUMATORE. Analogamente, la periferica CONSUMATORE non può processare il dato contenuto all'interno del buffer tampone prima che un nuovo dato sia stato generato dalla periferica PRODUTTORE.

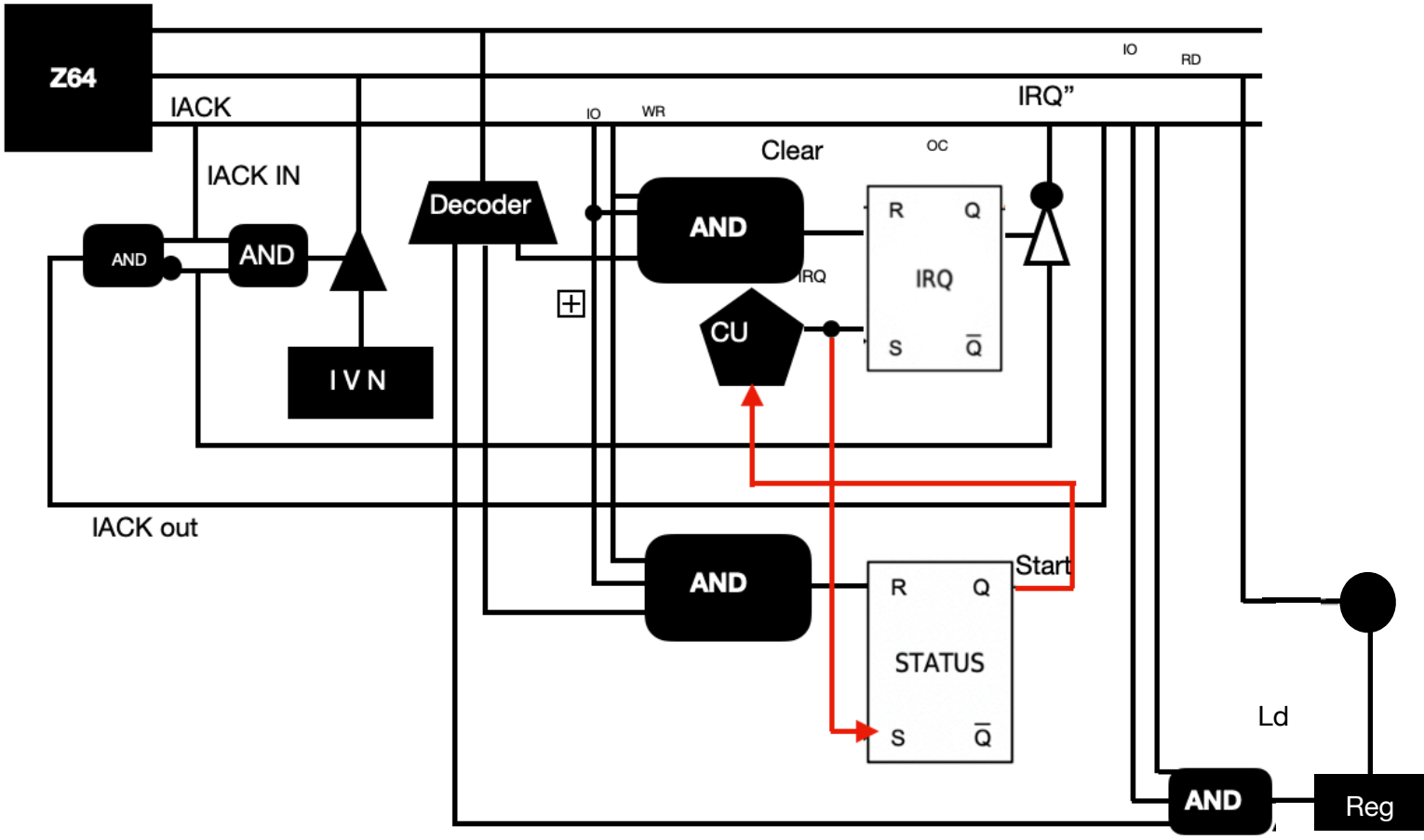
PRODUTTORE e CONSUMATORE lavorano con le interruzioni vettorizzate.

Svolgimento

Produttore - Input



Consumatore - Output



Come è possibile notare entrambe le interfacce possiedono il supporto per gestire le interruzioni vettorizzate.

Il flip flop **IRQ** indica al processore (leggendo opportunamente il valore di IRQ) se c'è stata o meno un'interruzione da parte della periferica - in modo particolare se è presente il valore IRQ = 1 (interruzione verificata) o IRQ = 0 (interruzione non verificata).

Il flip flop di **STATUS** notifica la CU del processore se i dati sono stati prodotti (interfaccia di input) o consumati (interfaccia di output) con successo.

Se il valore di status è 0 il trasferimento non è stato completato.

Se il valore di status è 1 il trasferimento è stato completato.

Il produttore vuole scrivere un dato nel buffer tampone.

Quindi il dispositivo deve essere avviato scrivendo 1 sul flip flop STATUS: i segnali di select, io e wr verranno impostati ad 1.

R = 1 di status.

Ora fin tanto che il dispositivo non ha finito di produrre i dati (input) l'output della macchina a stati della CU sarà pari a 0 (IRQ = 0) e di conseguenza il valore di S = 0 di status.

Quando il device ha completato la sua attività alzerà una bandierina e quindi IRQ = 1.

IRQ essendo un segnale che finisce in input al flip flop STATUS resetterà completamente il flip flop, quindi Q = 1 di status.

A questo punto dato che S = IRQ = 1, IRQ" = 0, e quindi interruzione verificata poiché ha finito di produrre dati.

I dati prodotti devono essere letti e perciò si abilita un segnale di io e rd che abilita un buffer three-state facendo fluire il contenuto del registro sul data bus della CPU.

N.b: solamente se S = 1 di status allora Q = 1 di status allora S = 1 di IRQ e di conseguenza IRQ = S = 1 con IRQ" = 0.

Il funzionamento del consumatore - ossia colui che deve processare i dati appena prodotti - è perfettamente analogo.

L'unico cambiamento è semplicemente la sostituzione del buffer three-state dal registro al data bus con un segnale di "ld" (load) che, se abilitato, abilita la consumazione e la conseguente scrittura dei dati prodotti nel registro di interfaccia.

Di seguito viene illustrato il codice.

```
.org 0x800
.data

        .equ $STATUS_CONSUMATORE, 0x0000
        .equ $STATUS_PRODUTTORE, 0x0001
        .equ $IRQ_PRODUTTORE, 0x0002
        .equ $IRQ_CONSUMATORE, 0x0003
        .equ $REG_PRODUTTORE, 0x0004
        .equ $REG_CONSUMATORE, 0x0005
        turno: .byte 0 #0 produttore, 1 consumatore
        buffer: .byte 0


.text
  main:
        outb %al, $STATUS_PRODUTTORE #Avvio della periferica produttore
        sti #abilito la ricezione delle richieste di interruzione. Imposto IF = 1
      .loop:
        btb $0, turno
        jnc .loop #se CF = 0 salta
        #questo è un loop fin tanto che non arriva un'interruzione da parte del produttore
        movb buffer, %al #sono il consumatore e prelevo il valore che il produttore aveva
inserito in buffer
        outb %al, $REG_CONSUMATORE #lo scrivo dentro il registro del consumatore
        outb %al, $STATUS_CONSUMATORE #notifico che il consumatore ha consumato i
dati e quindi start = 1 e IRQ" = 0 e viene eseguito il suo driver che resetta a zero il turno.
        jmp .loop
        hlt



   .driver 0 #PRODUTTORE
        push %rax #callee-save
        inb $REG_PRODUTTORE, %al #Leggo il valore dal registro del produttore
        outb %al, $IRQ_PRODUTTORE #cancello la causa di interruzione scrivendo 0 sul flip
flop.
        #omissione di movb $0, %al poiché stiamo parlando di DCC.
        movb %al, buffer #scrivo il valore precedentemente letto dal registro produttore nel
buffer
        addq $1, turno #passo al turno successivo
        pop %rax
        iret
    .driver 1 #CONSUMATORE
        outb %al, $IRQ_CONSUMATORE #cancello la causa di interruzione del
consumatore
        addq $1, turno #turno successivo
        iret
```