

Uno svantaggio degli schemi precedenti (interazione semi-sincrona) è che la CU del processore potrebbe rimanere in attesa indefinita per un tempo illimitato risultando inutilizzata. Per superare questo inconveniente è necessario modificare l'interfaccia con i dispositivi.

## BUSY WAITING

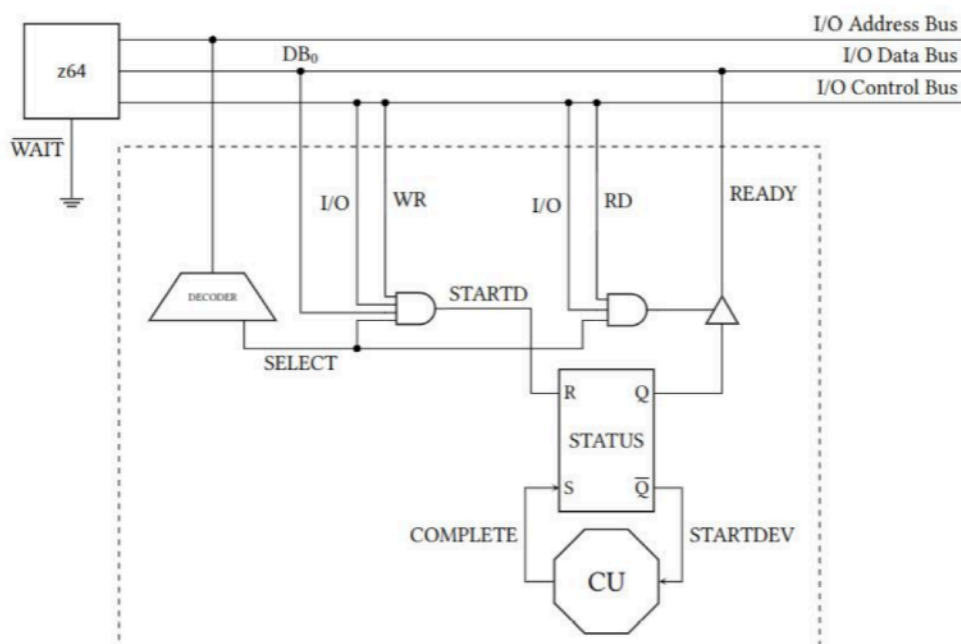
**Il processore chiederà ripetutamente (tramite software) al dispositivo se ha completato o meno la sua operazione.**

**Si utilizzerà il FLIP-FLOP di status che notificherà se una certa attività è stata completata.**

- 1. Il processore avvisa il dispositivo che vuole effettuare un trasferimento.**
- 2. Il processore verifica se il FLIP-FLOP STATUS è 1.**
- 3. Se è a 0, il trasferimento non è ancora completato: torna al punto 2.**
- 4. Il programma prosegue nel suo normale flusso.**

La CU del device leggerà opportunamente questo FLIP FLOP per notificare al processore se i dati sono stati prodotti (interfaccia di input) o consumati (interfaccia di output) con successo.

***Il processore però non esegue nessun'altra attività fino al completamento del trasferimento.***



**STARTD:** Questo segnale identifica la volontà del processore di avviare un protocollo di scambio dati con il device.

Nello specifico questo segnale resetta il FLIP FLOP di status.

**STARTDEV:** È il **buffering** del segnale **STARTD**.

Lo scopo di questo segnale (che diventa un ingresso della CU del dispositivo) è quello di generare una transizione di stato affinché l'operazione di interesse (produzione o consumazione dati) possa essere avviata.

**COMPLETE:** Questo è il segnale di controllo generato dalla CU del dispositivo. Questo segnale imposta il valore del FLIP FLOP STATUS ad 1, indicando che l'operazione richiesta dalla CPU è stata eseguita.

## **FUNZIONAMENTO GENERICO**

Il dispositivo deve consumare i dati che il processore intende mandargli.

Il processore scriverà l'indirizzo del device con cui vuole interagire,  $SELECT = 1$ ,  $I/O = 1$  e  $WR = 1$ .

Prendiamo anche il bit DB e lo settiamo ad 1.  
L'and vale 1,  $STARTD = 1$  e resetta il Flip flop.

A questo punto anche  $STARTDEV = 1 = Q$  e arriva questo segnale alla macchina a stati che implementa la CU e così inizierà a lavorare.

READY nel frattempo è 0, che è il valore di STATUS.

Quindi significa che l'operazione di consumazione (operazione generica, può essere anche produzione se parliamo di interfaccia di input) dei dati da parte del device non è completata.

Quando il device ha finito l'output della macchina a stati sarà il segnale di COMPLETE che setta  $S = 1$  e ci comunica che la periferica ha terminato di lavorare settando anche  $Q = 1$ .

Come fa il processore a sapere se il dispositivo ha finito di lavorare se il segnale di wait è finito a massa? Legge il valore del Flip flop di status.

Si prende il valore di Q e lo porta sul Data Bus, viene letto tramite un'operazione di IN dall'indirizzo che seleziona il Flip flop di status,  $I/O = 1$  e  $RD = 1$ , l'and dei tre segnali abilita il buffer-three state e il valore di Q arriva fino al processore.

Nota bene: questo schema funziona con entrambi i metodi quindi sia se dobbiamo scrivere dati sul processore sia se dobbiamo consumarne dal processore, i segnali in questa rappresentazione Busy waiting effettuando la gestione di questo approccio non sono i segnali di lettura o scrittura dei dati stessi contenuti nel registro del device.

Possiamo combinare le interfacce di input e di output con questo schema a Busy waiting.

---

*Se il flip flop è azzerato il processore ha attivato il dispositivo che però non ha ancora completato la sua attività.*

*Se il flip flop è impostato il dispositivo ha completato l'ultima attività richiesta dal processore.*

---

## **Come acquisire il valore del FLIP FLOP STATUS?**

### **SOFTWARE**

Il processore richiederà un'attività sui dispositivi ma sarà il software che si chiederà se un determinato dispositivo ha completato o meno qualche operazione.

# IN, OUT

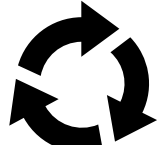
Le istruzioni di “**IN**” servono per richiedere la lettura di un dato *dal* dispositivo.  
Le istruzioni di “**OUT**” servono per richiedere la scrittura di un dato *sul* dispositivo.

L'unico registro che può essere usato per trasferire dati sia verso l'esterno che verso l'interno è **RAX** (inclusi tutti i suoi virtuali).

	SINTASSI	ISTRUZIONE	SEMANTICA
PARAMETRICA	<b>inX %dx, RAX</b>	Trasferimento in entrata da I/O parametrici.	Trasferisce i dati di dimensione X dal dispositivo distribuito sull'address I/O contenuto nel registro %DX.
ESPLICITA	<b>inX \$ioport, RAX</b>	Trasferimento in entrata dalla porta I/O esplicita.	Trasferisce i dati di dimensione X dal dispositivo distribuito sull'indirizzo I/O \$ioport.
PARAMETRICA	<b>outX RAX, %dx</b>	Trasferimento in uscita all'I/O parametrico.	Trasferisce i dati di dimensione X contenuti in RAX verso il dispositivo sull'indirizzo I/O contenuto nel registro %dx.
ESPLICITA	<b>outX RAX, \$ioport</b>	Trasferimento in uscita all'I/O esplicito.	Trasferisce i dati di dimensione X contenuti in RAX verso il dispositivo distribuito sull'indirizzo I/O \$ioport.

Nella sintassi parametrica si usa il registro DX come appoggio.  
Esso è un registro a 16 bit (word) che ci dice che in totale possiamo avere  $2^{16}$  porte di I/O.

**Buona Pratica:** Se non voglio scrivere l'indirizzo del dispositivo nell'istruzione ma vogliamo calcolarlo a RUN-TIME mentre il programma gira uso DX.



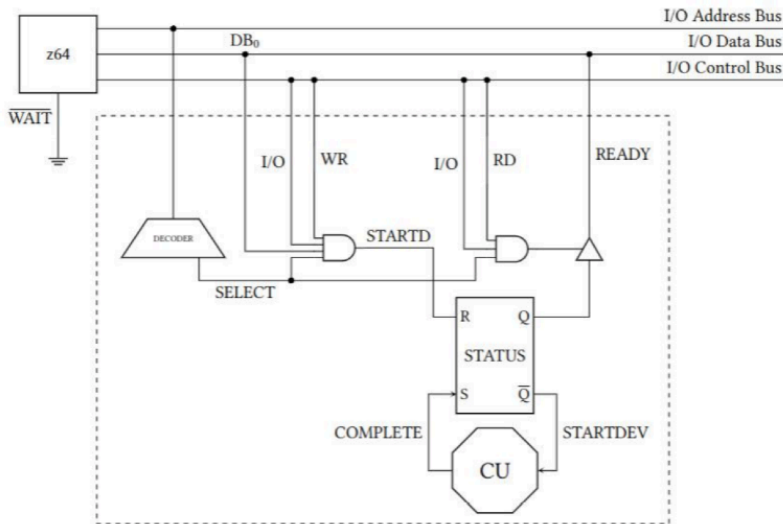
```
movb $1, %al
outb %al, $device

.bw:
inb $device, %al
btb $0, %al
jnc .bw
```

- L'unico modo per avviare il dispositivo è avere il valore “1” dentro %RAX, perciò scrivo un byte in %al (il suo registro virtuale).
- L'esecuzione della seconda istruzione scrive 1 sul Data Bus, abilita i segnali di I/O e WR, scrive l'I/O Port sull'address bus e così STARTD = 1 e resetto il Flip Flop di status.

In questo modo il device ora sta lavorando e il processore non sa quanto tempo ci impiegherà per processare (output) o per produrre dati (input).

Ora il processore prova ad abilitare I/O e RD con la terza istruzione in modo tale da poter leggere il valore del Flip Flop di status.  
Il valore del Flip Flop di status lo copia in AL.



In questa lettura viene letto 1 byte - ma in realtà status è un singolo bit - quindi con l'esecuzione della quarta istruzione "btw" copia il valore del bit all'interno del Carry Flag (CF).

In sostanza verifica se il bit numero zero partendo dal meno significativo è 1 o 0.

- Se CF = 0 (quindi il valore del Flip Flop status è 0) torna a ri-eseguire il codice [LOOP].
- Se CF = 1 (quindi il valore del Flip Flop status è 1) il dispositivo ha finito e posso proseguire nell'istruzione.

#### Problema:

- Il processore esegue lo stesso codice finché non è completato il trasferimento;
- Nonostante il processore non sia in stallo, non vengono effettuate attività utili.

#### Possibile soluzione:

- Interrogare altre periferiche e servire la prima disponibile.