

Tipi di interazione tra CPU e dispositivi

- **I/O programmato:** è il programma software che inizializza e governa le interazioni con i dispositivi per effettuare il trasferimento di dati.
↳ Si lascia la libertà di fare altre mentre un dispositivo sta eseguendo l'interazione col processore.
- **Su richiesta esterna:** il dispositivo richiede l'attenzione della CPU che esegue un driver per gestire il trasferimento dati.
↳ La CPU programma un dispositivo per fare qualche operazione ma dopo lo lascia lì e se ne dimentica, quando il device ha completato, esso
- **Gestite da processori dedicati (canali):** il processore demanda ad un coprocessore l'operazione di trasferimento dati.

I/O PROGRAMMATO

È IL SOFTWARE CHE GUIDA L'INTERAZIONE TRA I DISPOSITIVI E IL PROCESSORE!
IL PROCESSORE RICHIEDERÀ UN'ATTIVITÀ SUI DISPOSITIVI, MA SARÀ IL SOFTWARE CHE SI CHIEDERÀ SE UN DETERMINATO DISPOSITIVO HA COMPLETATO O NENDO UNA QUALCHE OPERAZIONE.

ABBIAMO NECESSITÀ DI INTRODURRE DELLE ISTRUZIONI DEDICATE: IN, OUT!

LE ISTRUZIONI DI IN SERVONO PER RICHIEDERE LA LETTURA DI UN DATO DAL DISPOSITIVO!
LE ISTRUZIONI DI OUT SERVONO PER ESEGUIRE LA SCRITTURA SUL DISPOSITIVO!

L'UNICO REGISTRO CHE PUÒ ESSERE USATO, SIA PER TRASFERIRE DATI VERSO L'ESTERNO CHE VERSO L'INTERNO, È RAX;

↳ INCLUSI TUTTI I VIRTUALI.

SI USA IL REGISTRO ACCUMULATORE RAX.

COME FACCIAMO A SPECIFICARE LA DESTINAZIONE O LA SORGENTE DI UN'OPERAZIONE DI INPUT E OUTPUT?

- **INDICAZIONE ESPLICITA:** → si usa Dx come registro d'appoggio (2nd PORT di I/O) (16 bit)
se non vogliamo scrivere l'indirizzo del dispositivo nello' istruzione, ma vogliamo calcolarlo a run-time mentre il programma gira, uso dX.
- **INDICAZIONE IMPLICITA:** → l'indirizzo di destinazione viene passato come costante,
è l'indirizzo da cui andiamo a leggere o l'indirizzo verso cui andiamo a scrivere.

Instruction	Syntax	Semantics
Inbound transfer from parametric I/O port	inX %dx, RAX	Transfer data of size X from the device deployed on the I/O address contained in the %dx register.
Inbound transfer from explicit I/O port	inX \$ioport, RAX	Transfer data of size X from the device deployed on the I/O address \$ioport.
Outbound transfer to parametric I/O	outX RAX, %dx	Transfer data of size X to the device deployed on the I/O address contained in the %dx register.
Outbound transfer to explicit I/O	outX RAX, \$ioport	Transfer data of size X to the device deployed on the I/O address \$ioport.

LO SCOPO È CHIEDERSI (ATO SOFTWARE, SE UN DISPOSITIVO HA TERMINATO O NENO, LA SUA ATTIVITÀ!

BUSY WAITING

= IL DISPOSITIVO VIENE AVVITATO E IL PROCESSORE ASPIRA CHE Venga TERMINATA LA RICHIESTA.

È SUB-OPTIMALE, NON DOBBIAMO ATTENDERE, ANZI DOBBIAMO fargli fare qualcosa IN PARALLELLO AL PROCESSORE, E LA TECNICA È IL POLLING!

BUSY WAITING

IL PROCESSORE PROGRAMMA UN DISPOSITIVO E Poi CONTINUA A CHIEDERE INSISTENTEMENTE AL DISPOSITIVO, SE HA TERMINATO DI FARE QUALCHE COSA.

“ HAI FINITO ? ” “ HAI FINITO ? ” “ HAI FINITO ? ”
 “ NO ” “ NO ” “ NO ” ...

LA RISPOSTA SARÀ “ NO ” PER UN SACCO DI TEMPO !

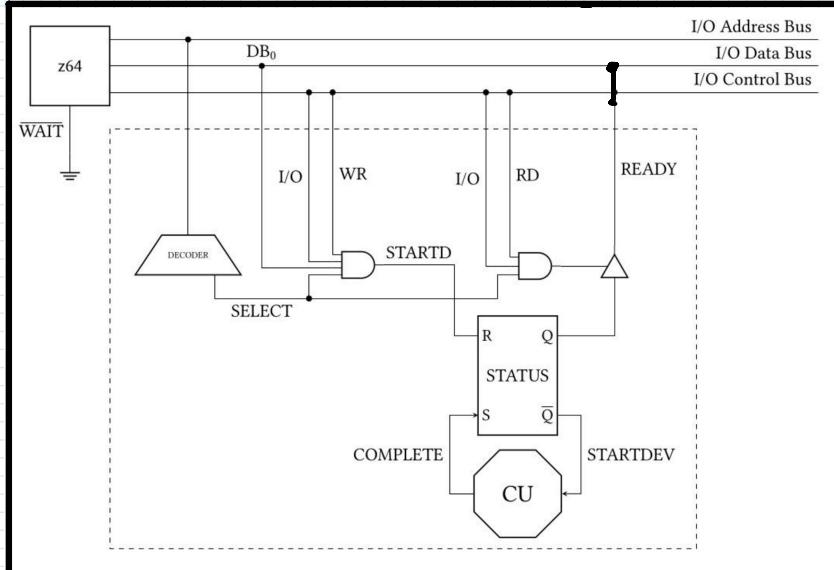
PRIMA O POI IL SOFTWARE PROSEGUITA PERCHÉ SI SPERA CHE IL DISPOSITIVO FINISCA DI FARE QUELLO CHE DEVE FARE !

COME REALIZZIAMO A SOFTWARE QUESTO MODO ?

IL PROCESSORE DEVE AVVISARE IL DISPOSITIVO CHE vuole EFFETTUARE UN TRASFERIMENTO, IN INPUT IN OUTPUT, IL PROCESSORE CICLERÀ IL BUSY WAITING, OVVERO LEGGERÀ IL VALORE CONTINUAMENTE DEL FLIP FLOP DI STATUS, CHE NOTIFICA CHE UNA CERTA ATTIVITÀ È COMPLETATA.

STATUS = 0, TRASFERIMENTO NON COMPLETATO.
 STATUS = 1, TRASFERIMENTO COMPLETATO.

RIPETIAMO FIN QUANDO STATUS VALE 1!



Ancora una volta abbiamo il flip flop di STATUS.
Ci permette di BUFFERIZZARE UNA QUALCHE RICHIESTA del processore.
IL PROCESSORE DEVE CHIEDERE AL DISPOSITIVO DI ATTARDI PER FARE QUALCOSA.

VOGLIAMO RESETTARE IL FLIP-FLOP.
PRENDO IL VALORE DEL REGISTRO DALL'ADDRESS BUS, SELECT VA A 1,
I/O e WR VANO A 1, PRENDIAMO

ANCHE IL BIT DAL DATA BUS DEL REGISTRO RAX,
SE TU SCRIVI 1 VERSO L'INDIRIZZO che SELEZIONA IL FLIP-FLOP DI STATUS, STAI RESETTANDO
IL FLIP-FLOP E STAI ATTARDI IL DISPOSITIVO!

SE HO DUE DISPOSITIVI SULLO STESSO BUS, COME SPEDIO A SOLO (AVENDO INTERFAZIE diverse) SE DEVO PROGRAMMARE UNA PIASTRA CHE L'ACCESA?
NON C'È UNO STANDARD, QUI PRODUTTORE PUÒ USARE L'INTERFACCIA HARWARE CHE VOLA!
QUESTI SONO I DRIVER;

↳ scrivere che sa come è fatta l'interfaccia hardware
del dispositivo, e la può programmare.

A QUESTO PUNTO L'AND VALE 1. START quando vale 1, resetta il flip flop!
Il flip flop vale 0, \bar{Q} vale 1, n segnale startdev avvia alla macchina a stati che implementa l'unità di controllo;

↳ che effettuerà una transizione chi si fa e inizia a lavorare.

Quando finisce, l'output della macchina a stati, sarà il segnale di complete che setta il flip flop, quindi la periferica ci vuole comunicare che ha terminato di lavorare!

Come fa il processore a sapere se il dispositivo ha finito di lavorare se il segnale di WAIT è già tornato a massa? Lo interroga: "hai finito?".

Questo significa leggere il valore del flip flop di status;

Quindi io posso prendermi il valore di Q e portarlo sul data bus, lo leggo con RD
dall'indirizzo che seleziona il flip flop di status, e l'and di quei 3 segnali abilita il buffer-three state e invia il valore del flip flop attuale fino al processore.

Questa è la lettura!

QUAL'È IL SOFTWARE CHE IMPIENTA TUTTO QUESTO ??

```
movb $1, %al  
outb %al, $device  
.bw:  
inb $device, %al  
btb $0, %al  
jnc .bw
```

All'inizio devo avviare il mio dispositivo, e per fare questo abbiamo dunque che devo SCRIVERE 1, ma l'unico modo per scrivere 1 è avere 1 dentro RX, devo leggere un solo bit, posso scrivere 1 Byte.

A questo punto lo mando verso il flip flop di status, l'esecuzione della seconda istruzione ABILITA i segnali di I/O WRITE, scende l'I/O port sull'address bus e scrive 1 sul data bus, STARTD vale 1, ho RESETTATO il Flip Flop;

Sarà l'esecuzione di questa istruzione, se il dispositivo sta lavorando, quanto ci metterà? Non so, ci metterà un po' di tempo;

Ora leggo il valore del flip flop di status, abitro I/O e RD, passo l'Address bus e leggo il valore del flip flop status, e me lo copio all'interno di AL.

In questa lettura ho letto un byte, ma in realtà status è un bit, quindi vado a verificare il valore del bit meno significativo, per questo uso l'istruzione di **BIT TEST (BT)**, che verifica se il bit numero zero partendo dal meno significativo È 1 o NO!

Copia il valore del bit significativo all'interno del carry flag.

SE CARRY FLAG VALE 0 SIGNIFICA che il dispositivo NON HA FINITO perché ha STATUS=0, SE CF=1, STATUS=1, ha finito e posso proseguire l'esecuzione;

L'ULTIMA ISTRUZIONE dice, SE IL CARRY vale 0, loop. E RILEGGO NUOVAMENTE IL VALORE DI STATUS, e vedo che STATUS vale 1.

HO IMPLEMENTATO A SOFTWARE quello che prima ho implementato a Firmware;

- Problema: il processore esegue lo stesso codice finché non è completato il trasferimento
- Nonostante il processore non sia in *stall*, non vengono effettuate attività utili
- Possibile soluzione: interrogare altre periferiche e servire la prima disponibile

POLLING

Anziché chiedere se un singolo dispositivo ha terminato, comincia a chiedere ad altri dispositivi se hanno terminato, uno dopo l'altro.

L'IDEA PUÒ ESSERE DI LEGGERE IL FLIP FLOP DI STATUS DEL PRIMO DISPOSITIVO, A QUESTO PUNTO TESTO IL BIT MENO SIGNIFICATIVO, SE VALE 1, OSSIA IL CF=1, SIGNIFICA CHE QUEL DISPOSITIVO È PRONTO E POSSER SALCARE IN QUALCHE ALTRO PUNTO DEL CODICE IN CUI INTERAGISCE SOLO CON QUEL DISPOSITIVO, SE ERA UN DEVICE DI INPUT SIGNIFICA CHE È PRONTO IL DATO CHE DOVREVA ANALIZZARE E LEGGERE, SE È DI OUTPUT AVRÀ FINITO DI PROCESSARE I DATI CHE GLI HA MANDATO.

Quindi quando ho finito di eseguire l'attività del singolo, ritorno al ciclo di polling.

E mi richiedo: "È PRONTO IL PRIMO?" NO
"È PRONTO IL SUCCESSIVO?"

Se ho tanti dispositivi, chieder ripetutamente se uno di questi è pronto, e aspetto che una mi risponda.

È STILE AL BUSY WAITING! MA NON ASPIRO PIÙ CHE UN SINGOLO DISPOSITIVO FINISCA DI PROCESSARE I DATI.
ANCHE IL POLLING È SUBOTTIMALE, PERCHÉ DOVO SEMPRE A CHIEDERE SE I MIEI DISPOSITIVI HANNO FINITO.

- Problemi: il processore deve continuamente testare il valore di STATUS
- Codice difficile da manutenere: cosa succede se si aggiunge un dispositivo?
- Rischio di starvation: le ultime periferiche possono non essere servite mai
- Difficile gestire le priorità

NOI VOGLIAMO CHE LA CPU FAGLIA PROPRIO ALTRO, DEVE ESSERE IL DISPOSITIVO A DIRE "HEY HO FINITO", E NON VOGLIAMO SCRIVERE MEZZA RIGA DI CODICE!

NON DOVO CHIEDERE CONTINUAMENTE SE HANNO FINITO!

È IL DISPOSITIVO CHE ALZA LA BANDIERINA QUANDO È PRONTO;

↳ questa tecnica richiede una modifica a livello hardware e si chiama: TECNICA BASATA SULLE INTERRUZIONI.

```
.poll :  
    movw $STATUS_DEV1, %dx  
    inb $STATUS_DEV1, %al  
    btb $0, %al  
    jc .dev1  
    inb $STATUS_DEV2, %al  
    btb $0, %al  
    jc .dev2  
    # ...  
    jmp .poll  
  
.devX:  
    # ...  
    jmp .poll
```