Lezione 37 Input e output

giovedì 7 marzo 2024 11:18

Vediamo ora quanto impatto ha l'utilizzo delle periferiche esterne sul nostro programma (ricordiamo che incremento delle prestazioni sono del 60%, mentre il ritardo dei componenti fisici si aggira intorno al 10%) : vediamo la **legge di Ahmdal** : quanto inserendo un processore più veloce , permettere di avere un incremento delle prestazioni :

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

Dove f è la parte di attività passata in processamento dalla cpu , 1-f quella dell' interazione con dispositivi , mentre k è il fattore di volte che voglio il processore più veloce.

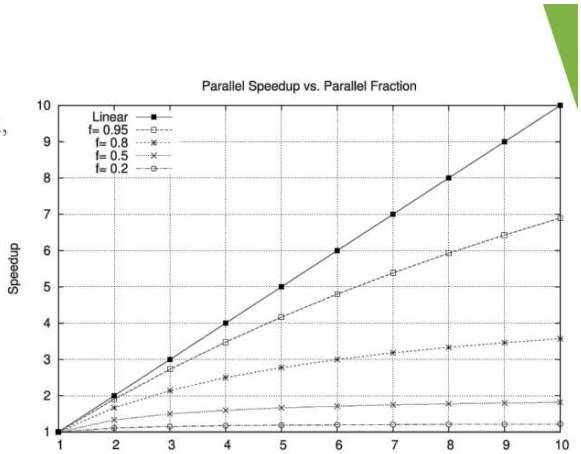
Vediamo un esempio :

I dispositivi di I/O determinano un collo di bottiglia prestazionale

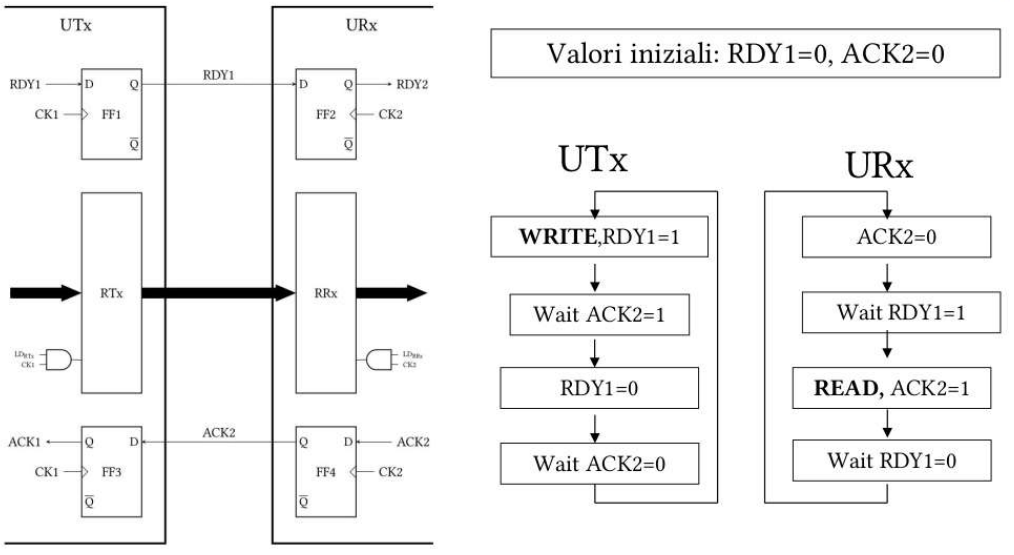
Anche se la parte f è elevata, lo speedup sarà limitato dall'I/O

Ad esempio, se $f = 80\%$:

$$\lim_{K\to\infty} S = \frac{1}{1-0.80} = 5$$

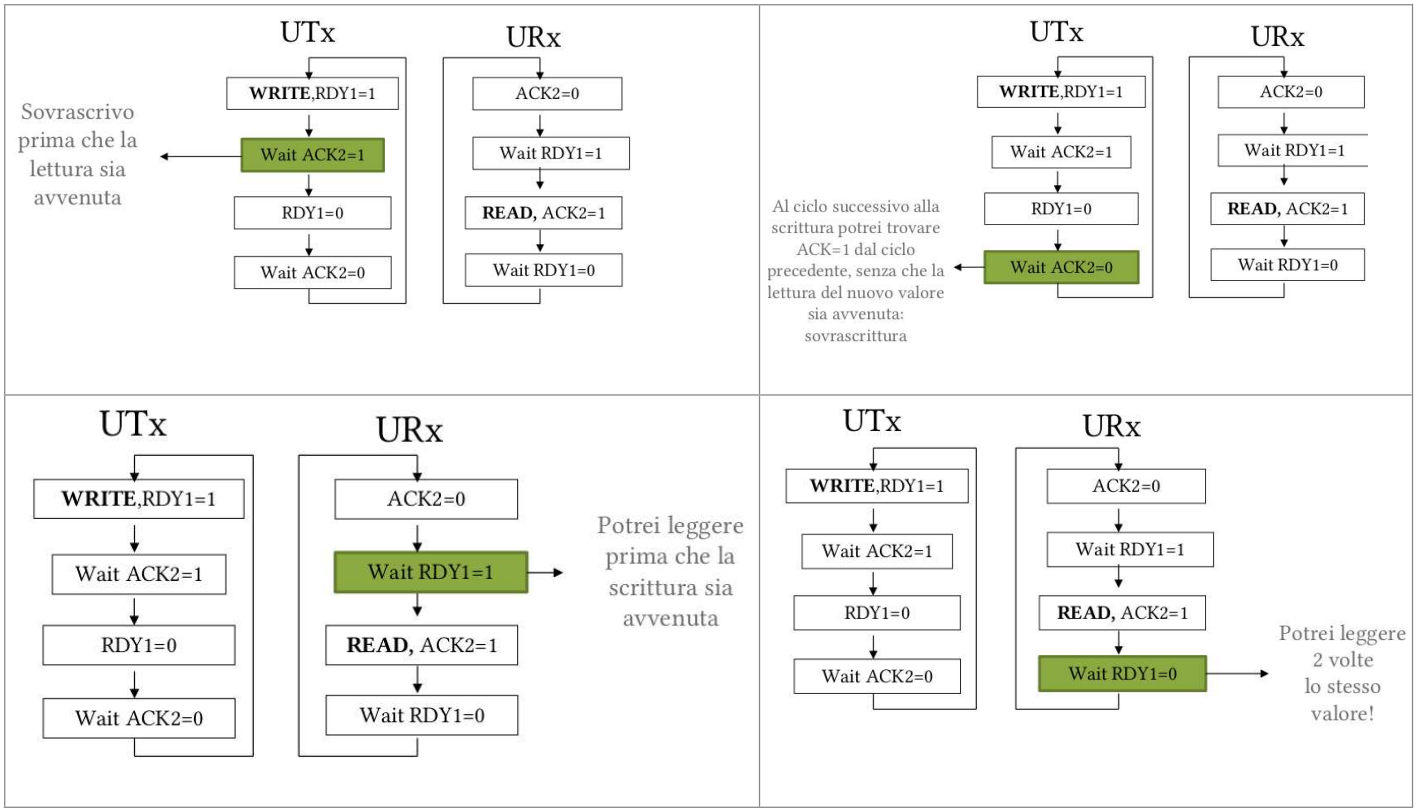


Come facciamo però in realtà a fare comunicare 2 o più dispositivi? Ci deve essere sia un canale di comunicazione , che la stessa velocità : quindi in questo caso si parla PROTOCOLLO DI HANDSHAKING : innanzitutto si "presentano" i dispositivi , poi c'è lo scambio di informazioni per far viaggiare i dati ad una stessa velocità (stesso clock): quindi si deve cercare di mantenere dati in modo stabile . Quindi viste le differenze di velocità e la necessità di mantenerli stabili, vi si devono usare dei registri tampone . Vediamo il circuito :

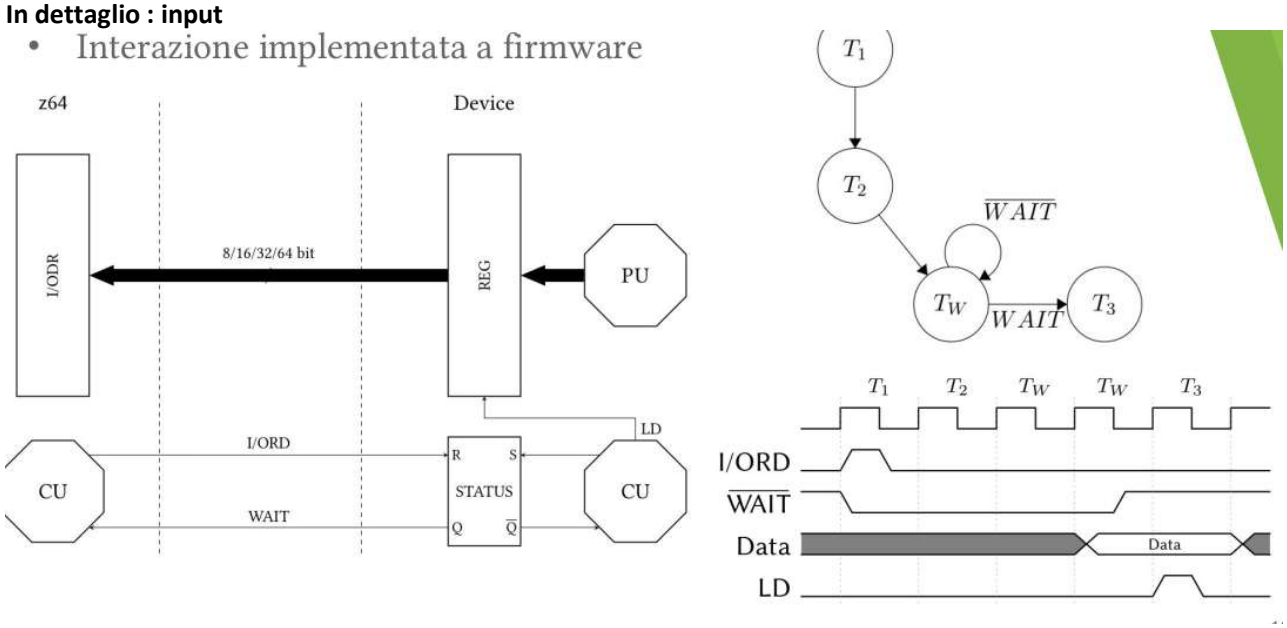


Focalizziamoci ora sui FF : ci basta un ulteriore bit per sapere se c'è un dato da leggere, mente il secondo bit serve a dire " ho finito di leggere". **Anche questi due segnali devono essere bufferizzati.**

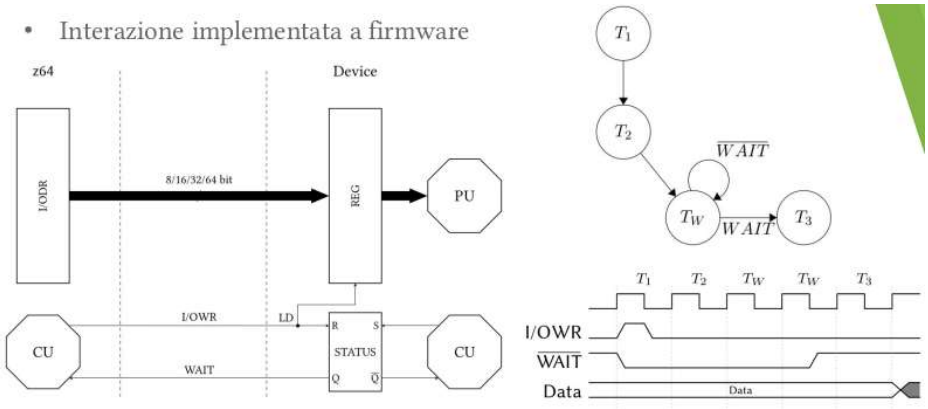
Quindi usando questi due registri tampone permettono inoltre la comunicazione tra registri e FF , in quanto hanno velocità lettura e scrittura differenti. Quindi in dettaglio :



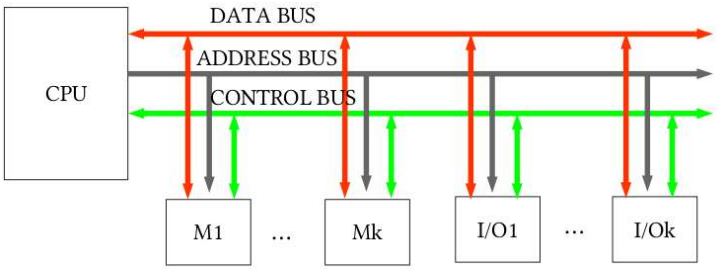
In dettaglio : input



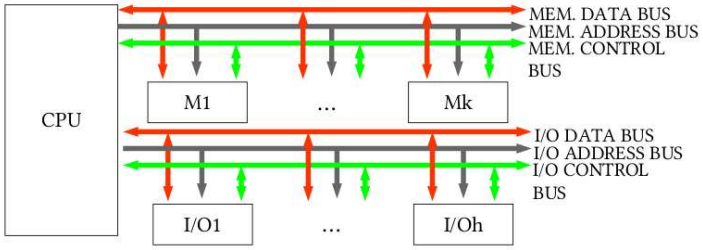
Innanzitutto ricordiamoci che il processore è molto più veloce dei dispositivi, quindi dovrà attendere che producano dati , mettendosi in attesa. Quindi si ha un solo FF (chiamato stato) che utilizza il processore per chiedere al dispositivo di mandare i dati. Quindi l'unita di controllo (CU) invia il segnale io/rd (lettura da input/output) , il quale segnale viene scritto nel ff contenuto nel processore. Dato che il dato è cambiato , ed il processore quindi ha resettato il ff (segnale reset) , inizia a produrre i dati , i quali verranno mandati sul registro di interfaccia. Dati pronti e scritti quando la CU abilita il segnale di io/rd. Attenzione anche al segnale di wait (aspetta) : vale 0 fin tanto che il processore deve aspettare finche' l'UC(unità di controllo) scrive i dati sul registro , vale 1 (settato) quando i dati sono stati scritti. Se nel ff si ha il set(segnale ad 1) anche il wait cambia: si ha quindi transizione di stato. **Andiamo ora a vedere l'output** :



Cambia il modo in cui andiamo a scrivere i dati nel registro. Il processore bufferizza i dati nel registro di interfaccia, manda il segnale di io/wrt al ff . Attenzione al collegamento del segnale LD(dove è). Per il resto è analogo. **Questo schema è quello che si usa, ma dal punto vista prestazionale fa schiantare il processore : spreca infinità di ciclo ci clock** . Andiamo a vedere come generalizzare questa connessione : si hanno dei bus di interconnessione , i quali viaggiano all'interno della scheda madre , per andare a connettere i vari pezzi :



Notiamo che in questa configurazione , c'è un solo bus condiviso con la memoria (bus di sistema) , ed è efficiente perché riduce il numero di fili , ma vediamo che il bus di indirizzi viene usato dalla memoria e dai dispositivi, non si sa con chi parlo : sia hanno spazi di indirizzamento condivisi (dispositivi rubano indirizzi alla memoria): configurazione **memory mapped.** Vedasi istruzione mov. In alternativa : utilizzo altro bus : un bus per la memoria ed un altro dedicato per i dispositivi i/o :

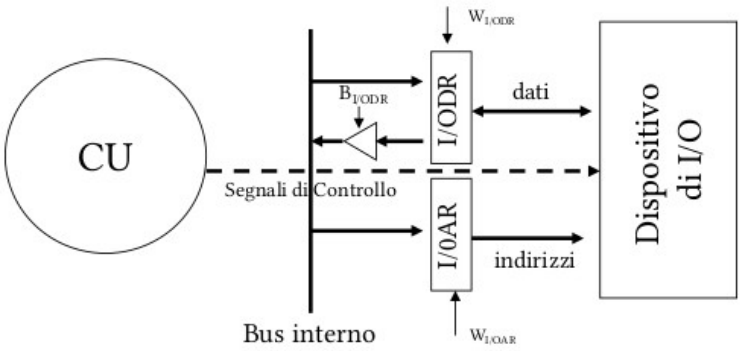


Ha spazi indirizzamento diversi : possibile utilizzo tutti e 64 bit, ma avendo due bus diversi , ho dei registri di interfaccia per ognuno di questi bus. Vedasi registri MAR e MDR . Quindi per quanto detto ora dobbiamo modificare l'interfaccia dello z-64 , per consentire di operare su più bus: quindi si parte da questa :

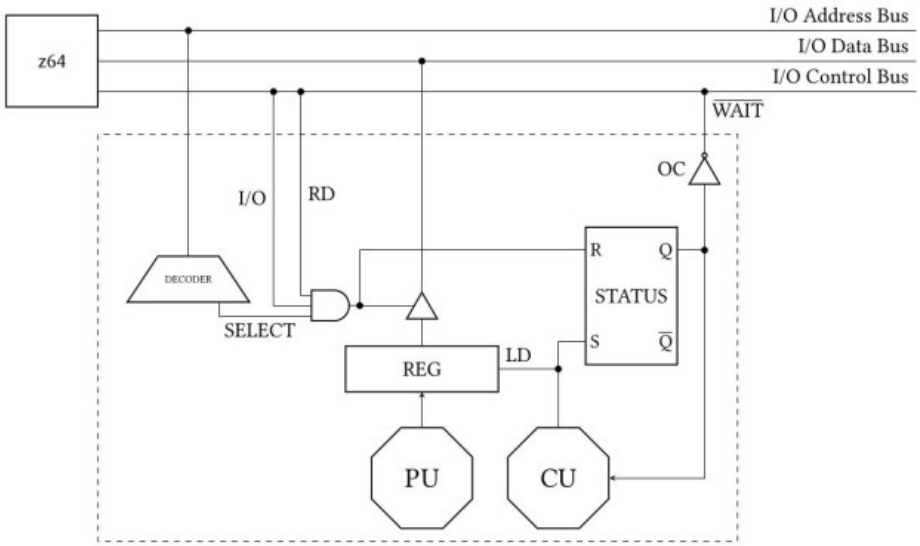
Registro Dati (I/ODR)

Registro Indirizzo (I/OAR)

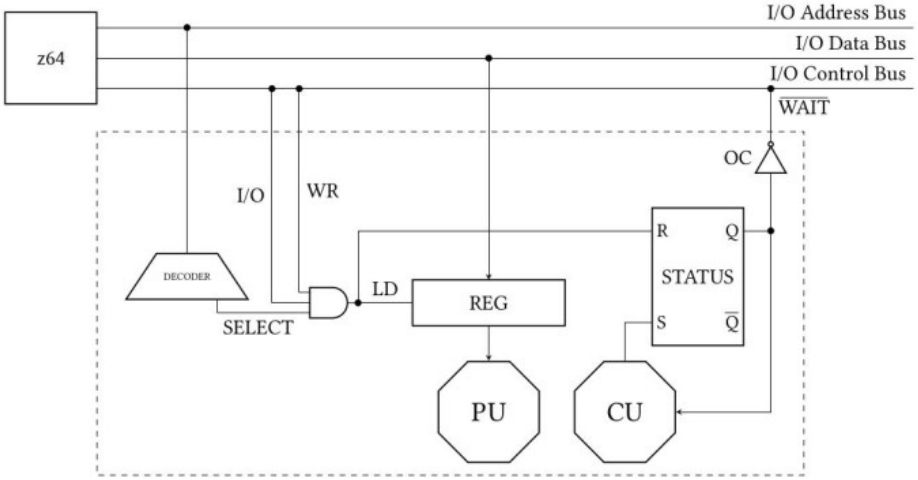
Segnali di Controllo (I/O, RD, WR, ...)



e si arriva a questa **(per input)** :



In dettaglio : andiamo a vedere il decoder : da un ingresso di 64 bit, attiva una sola uscita , avendo cosi un solo dispositivo attivo. Quindi dopo questa fase i segnali i/o e rd arrivano a tutti i dispositivi , ma avendo la combinazione in and con indirizzo di un solo dispositivo , poi il processore abilita il buffer tri state per scrivere i dati sul data bus ed evitare che il ff si bruci (ingresso q). Per quanto riguarda anche il wait , anche qui altro buffer tri state per evitare che il segnale q non si propaghi ad altri ff, usando la tecnologia di Open-collector : uso il not per calcolare l'or cablato di tutti i dati che arrivano dai dispositivi di input : il processore si mette in wait finche' almeno un dispositivo che produce dati. **Nel caso dell'output** :



Il processore mette dati sul data bus: unico dispositivo determinato dal decoder , permettendo cosi' il flusso dei dati ed il dispositivo può incominciare a processarli. Analogamente tecnologia OC del ff . Quindi per interazione multi-dispositivo si usa una linea di abilitazione alla quale risponde uno ed uno solo dispositivo alla volta.