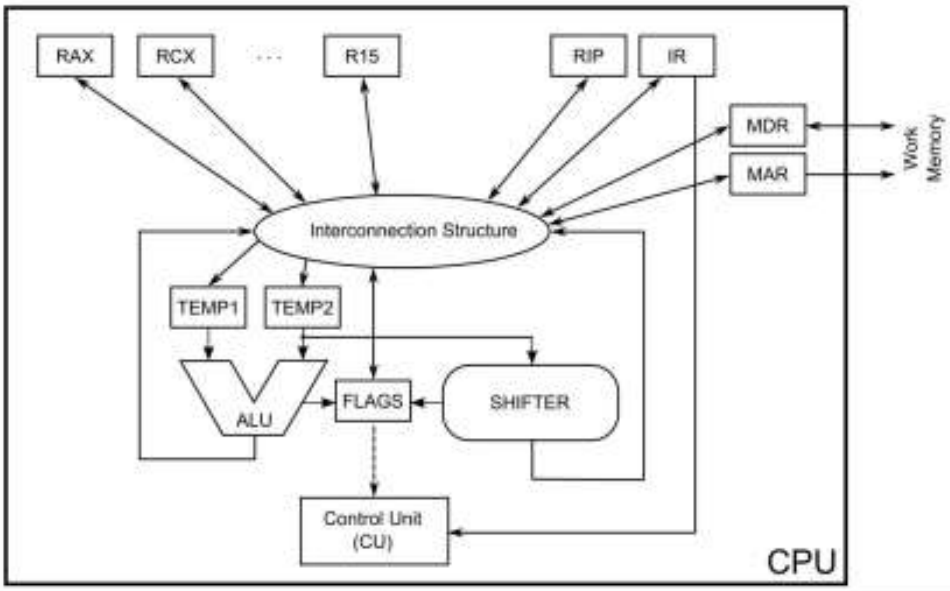
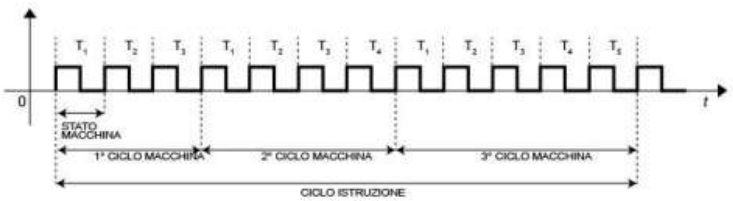
Lezione 16 Z64 2

giovedì 2 novembre 2023 17:51

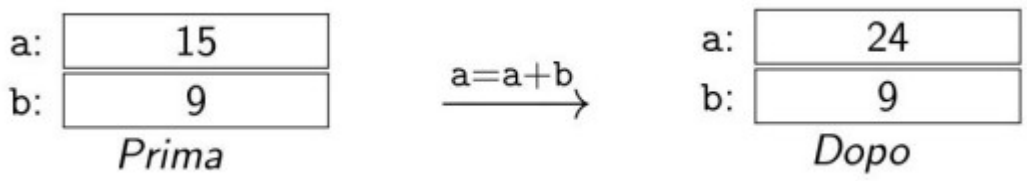


Questo modello è abbastanza datato in quanto è un modello **uni ciclo** : organizzazione interna dei processori nel quale ogni istruzione viene eseguita è un colpo di clock, il quale viene preso minimo in base al critical path (per la stabilizzazione) e se operazioni differenti non posso riciclare hw , quindi aumenta il numero di componenti , mentre per quelle **multi ciclo** : rompo il circuito in più parti, accorcio il tempo di clock : un'istruzione è eseguita in più colpi di clock , inoltre le componenti possono essere riusate tra un ciclo ed il successivo . Con questo modello si può avere il riciclo dell'hw . Da notare che nella fase di esecuzione , si eseguono più istruzioni : quindi la fase di execute si spezza in sotto-fasi , quindi per eseguire un'istruzione :

- *Ciclo istruzione*: intervallo temporale necessario ad eseguire una istruzione nella sua interezza
- *Ciclo macchina*: intervallo temporale necessario ad eseguire una fase (fetch, decode, execute)
  - A seconda del tipo di istruzione, possono essere necessari un numero diverso di cicli macchina (es: più accessi in memoria)
- *Stato macchina*: periodo di tempo necessario per stabilizzare la rete dell'unità di calcolo (corrispondente al clock)



Notiamo che non tutte le istruzioni hanno lo stesso ciclo macchina. Vediamo ora la semantica di un'operazione :



Dove a e b sono i registri : li consideriamo come due scatolette . **Quindi prima dobbiamo capire dove sono memorizzati, decidere dove scrivere il risultato , decidere quali istruzioni svolgere .**

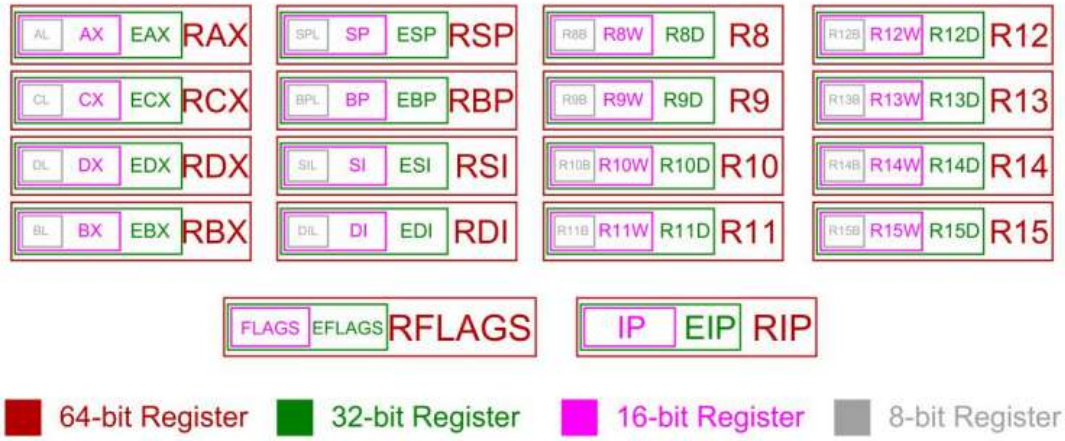
Il formato è il seguente :

MOVs <sorgente>, <destinazione>

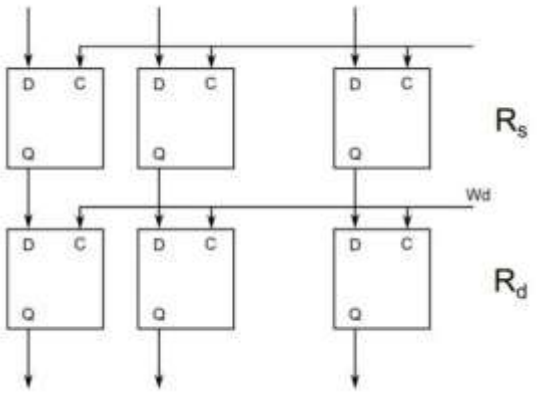
s può essere B, W, L, Q-

Dove byte, word, long-word, quad-word : 8-16-32-64

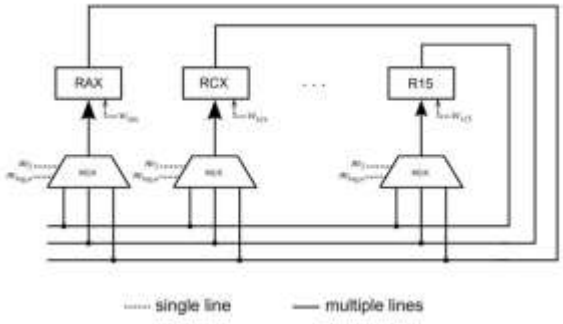
In dettaglio (general purpose) :



Possiamo virtualizzare i registri : uso i registri di più piccoli (da 64 si va 32 ecc ecc) : attenzione che si corrompono i registri . L=low meno significativi , E= extended 32 bit . Da R8 a R15 sono quelli di amd , che intel li ha presi : registri veramente general purpose. Quelli in basso partono da 16 bit . Come sono realmente realizzati questi registri? Attraverso dei FF collegati tra loro , attraverso fili (il valore viene memorizzato ad ogni colpo di clock) :

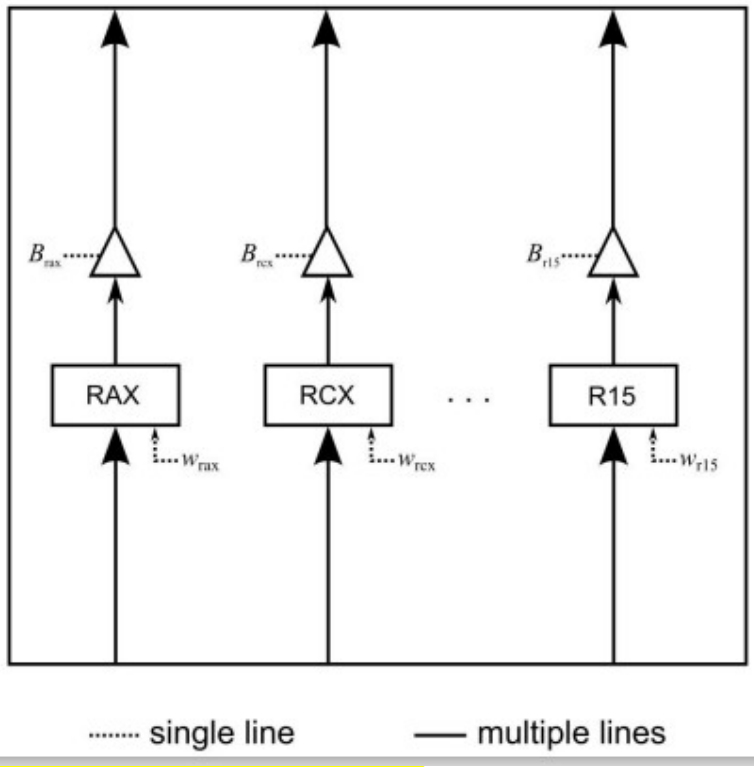


Ma notiamo che se il numero dei FF cresce , si hanno molti collegamenti , quindi possibile aumento complessità (64 fili per ogni registro e verso tutti i registri). In alternativa a questa connessione punto-punto , lo implementiamo cosi : attraverso uso dei mux (in ingresso scegli uno sola configurazione) , cosi da 64 fili per ogni registro se ne ha uno solo :



Notiamo che per ogni registro si hanno dei segnali di controllo, i quali permettono o meno la scrittura del dato nel registro , basta che siano abilitati (segnalo Wrax). Riassumendo : **1 canale**

di 64 bit per ogni registro, ad ingresso un mux che sceglie l'uscita corrispondente ed un segnale di abilitazione di lettura/scrittura. Questi segnali vengono generati dall'unità di controllo. Problema con questo schema : costo eccessivo dei mux e delle linee di interconnessione . Si può migliorare ? Si : interconnessione tramite bus : quindi i registri possono leggere e scrivere su stesso collegamento : possibile sia ricezione che trasmissione , ma c'è un controllo che permette ad un unico registro di trasmettere i dati usando due bit di controllo : Write enable per scrivere e buffer-tri state che si comporta da interruttore :



Avendo cosi solo 64 fili di interconnessione. Si deve separare il momento della lettura da quello della scrittura , per evitare che i dati non sono stabili. Torniamo ora ai principali due registri : IR e RIP : il primo mantiene logicamente informazione su ultima istruzione eseguita (memorizzo indirizzo prossima istruzione) , la quale è ad indirizzo prossimo : architettura Von Neumann funziona sulla sequenzialità delle istruzioni , a meno che non gli si chiede di saltare ad istruzioni "lontana". Mentre per quello che riguarda il secondo mantiene in memoria l'indirizzo della prossima istruzione da eseguire.