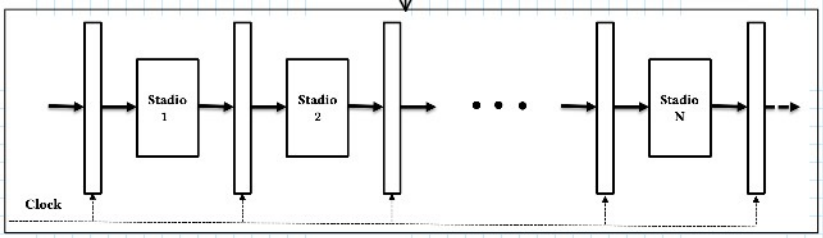
Schema PIPELINE

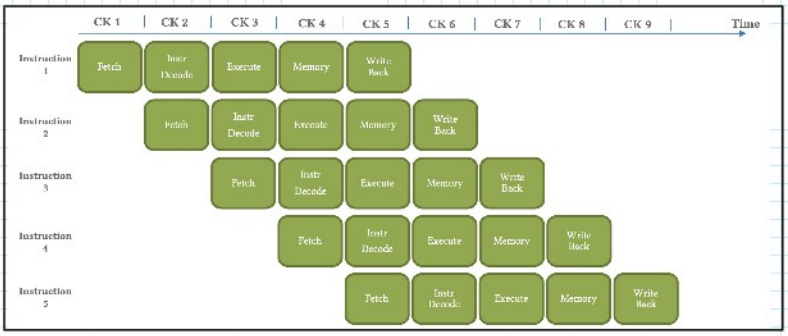
lunedì 27 febbraio 2023 14:50

Schema di principio delle architetture pipeline

- Gli stadi devono essere macchine indipendenti



COMPLETERÒ L'ESECUZIONE DELL'ISTRUZIONE IN UN SOLO COLPO DI CLOCK.



TIRO FUORI UN'ISTRUZIONE COMPLETATA, VIRTUALMENTE OGNI COLPO DI CLOCK!

CIASCUNO STADIO OPERA IN MANIERA INDIPENDENTE DUNQ DALL'ALTRO.

ORA, SE IO HO DIFFERENTI CIRCUITI COMBINATORI / STADI, CHE OPERANO SU ISTRUZIONI DIFFERENTI, COME FACCIO A RENDERLI INDIPENDENTI? REGISTRI TAMPONE.

OSSIA DEVO REALIZZARE DEI CIRCUITI COMBINATORI IN CUI L'INPUT DI CIASCUNO STADIO RIMANGA STABILE DURANTE TUTTA L'ESECUZIONE DI QUELLO SPECIFICO STADIO.

QUINDI TRA UNO STADIO E L'ALTRO DEVO INSERIRE DEI REGISTRI TAMPONE CHE MANTENGANO STABILE L'INPUT A QUELLO SPECIFICO STADIO, CHE È IL REQUISITO STANDARD PER TUTTE LE RETI COMBINATORIE.

IL RISULTATO DELL'ESECUZIONE DI UNO SPECIFICO STADIO VIENE MEMORIZZATO NEL REGISTRO TAMPONE CHE LO DIVIDE DALLO STADIO SUCCESSIVO, QUINDI IL RISULTATO DI QUESTA COMPUTAZIONE PARZIALE VERRÀ PASSATO COME INPUT ALLA COMPUTAZIONE PARZIALE SUCCESSIVA.

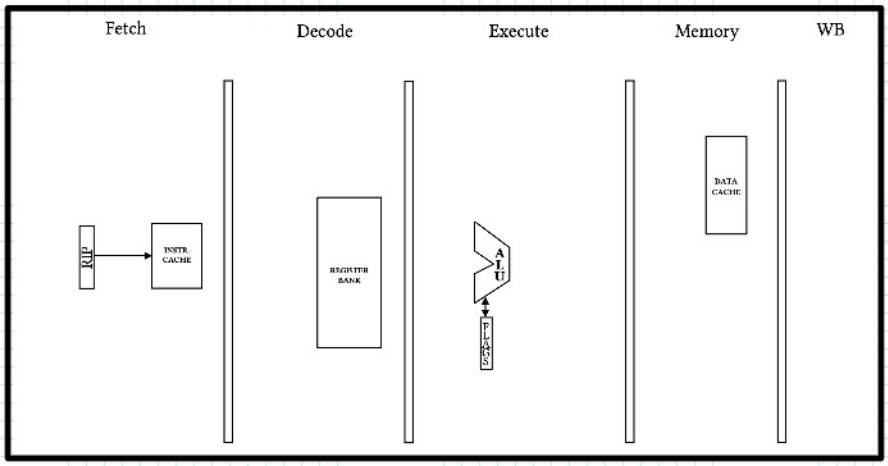
QUESTI REGISTRI TAMPONE, NELLA PIPELINE, SI CHIAMANO REGISTRI DI PIPELINE.

POSSO REPLICARE PIÙ COMPONENTI HARDWARE (ESEMPIO SOMMATORE) IN STADI DIFFERENTI.

IN FIN DEI CONTI OGNI STADIO È INDIPENDENTE E IN OGNI STADIO ESEGUO IN PARALLELO PEZZI DIVERSI DI ISTRUZIONE.

SE HO UNA ALU CHE STA ESEGUENDO UNA SOMMA NELLA FASE DI EXECUTE, LA FASE DI FETCH AVRÀ BISOGNO DI UN SOMMATORE DIFFERENTE, QUINDI EVITO DI PRENDERE LA ALU.

QUANDO L'HARDWARE DELLO STADIO ASSOCIATO ALLA FASE DI FETCH È LIBERO, POSSO ESEGUIRE IL FETCH DELL'ISTRUZIONE SUCCESSIVA PER PARALLELIZZARE.



ELEMENTI FONDAMENTALI:

- PROGRAM COUNTER: Lo mettiamo nella fase di fetch, esso ci permette di sapere qual'è la prossima istruzione che vogliamo andare ad eseguire.

DA DOVE LA PRENDIAMO LA NOSTRA ISTRUZIONE? DALLA memoria, PERÒ ABBIAMO DETTO CHE IL PROCESSORE NON PARLA DIRETTAMENTE CON LA MEMORIA;

IL PROCESSORE PARLA CON LA CACHE. LA CACHE DI PRIMO LIVELLO È DIVISA IN CACHE ISTRUZIONI E CACHE DATI.

IMPLEMENTARE L'ARCHITETTURA HARVARD NEL PRIMO LIVELLO DI CACHE, È UNO STEP FONDAMENTALE PER POTER REALIZZARE UN'ADEGUATA architettura a PIPELINE.
PERCHÈ NELLO STADIO DI FETCH IL MIO PROGRAM COUNTER MI PERMETTERÀ DI INDIRIZZARE L'ISTRUZIONE SUCCESSIVA dall' INSTRUCTION cache.
I DATI PROVERRANNO/VERRANNO SCRITTI dalla/sulla data cache.

Poichè SIAMO IN due STADI differenti NON POSSIAMO condividere L'HARDWARE, quindi LA NECESSITÀ di differenziare L'INSTRUCTION cache dal data cache, NASCE dal fatto CHE IN UN'ARCHITETTURA PIPELINE IO NON POSSO fare IL fetch IN parallelo ad una SCRITTURA dati, dividendo IN due la cache ho due componenti hardware che possono OPERARE IN PARALLELO, perchè ALTRIMENTI IL BUS sarebbe uno, IL CONTROLLORE della cache sarebbe uno solo.

Quindi effettivamente riesco a fare il fetch mentre leggo/scrivo dalla memoria.

NELLA FASE di decode leggiamo IL CONTENUTO dei REGISTRI, questo lo facciamo per comodità perchè IN questa fase posso INSERIRE IL BANCO dei REGISTRI, identico all' organizzazione che ABBIAMO VISTO NELL'ARCHITETTURA a multiciclo, che MI SERVIRANNO PER SUPPORTARE quell'ISTRUZIONE.

NELLA FASE DI EXECUTE devo SUPPORTARE L'ESECUZIONE della mia ISTRUZIONE, NON PUÒ MANCARE LA ALU e NON PUÒ mancare IL REGISTRO flags

NELLA FASE di memory IN CUI SCRIVIAMO/Leggiamo valori della memoria, ABBIAMO detto che INSERIAMO LA DATA CACHE, Possiamo leggere e SCRIVERE dati da e VERSO la memoria;

NELLO STADIO DI WRITE BACK NON ABBIAMO NECESSITÀ di HARDWARE PARTICOLARE, perchè VEDREMO poi cosa DOBBIAMO ANDARCI ad aggiungere.