Simulatore di pipelining

Un simulatore grafico di semplice utilizzo per lo studio del comportamento e dello stato interno di un processore dotato di pipeline, un passo alla volta.

Il pipelining in un processore

Il processore è un componente hardware fondamentale del calcolatore. Esso ha il compito di eseguire una serie di istruzioni , prodotte sulla base di un instruction set. In generale, il processore è costituito da varie componenti che svolgono compiti differenti. Un processore scalare a 5 stadi, in condizioni ottimali, è capace di cominciare e terminare l'esecuzione di un'istruzione in 5 cicli di clock.

Un processore con architettura a pipeline sfrutta la propria suddivisione interna sulla base di un concetto analogo a quello della catena di montaggio nell'industria meccanica. Piuttosto che riservare l'intera catena ad un solo oggetto, essa viene spartita tra più oggetti in stadi differenti di lavorazione. Ciò permette , in linea di principio , un throughput di 1 istruzione per ciclo di clock.

Struttura di un processore con pipeline

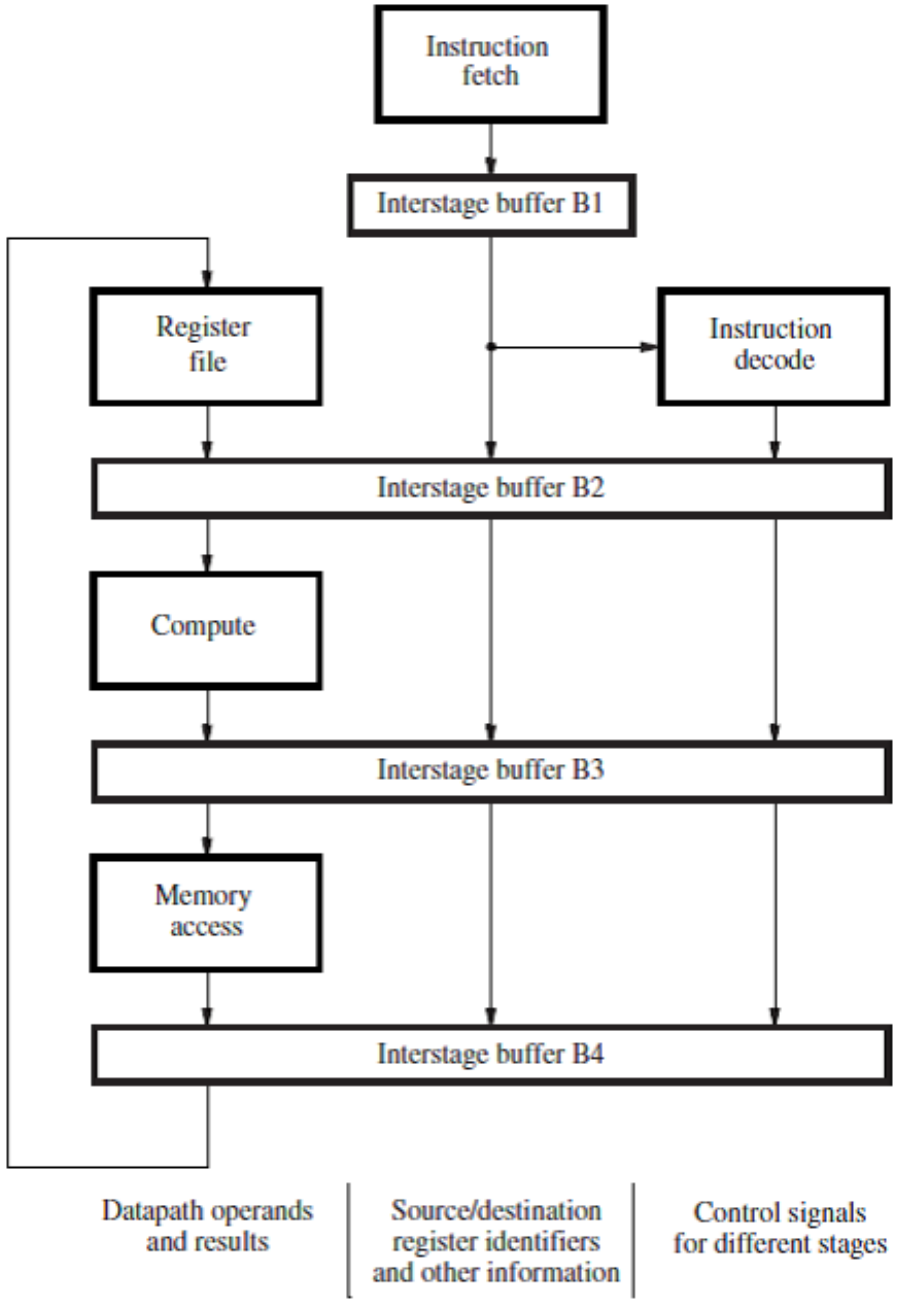


Figure 6.2 A five-stage pipeline.

Il programma simula il comportamento di un processore a cinque stadi con pipeline. Ogni esecuzione di istruzione in esso viene suddivisa in cinque fasi : prelievo(fetch), decodifica(decode), esecuzione(compute), accesso alla memoria(memory access) e scrittura dei registri(write back).

Tra una fase e l'altra dei buffer interstadi conservano le informazioni importanti per un'istruzione durante il suo avanzamento nella pipeline, come il valore di registri interstadi (RA, RB, RZ, RM, RY) e dei registri PC-temp e IR.

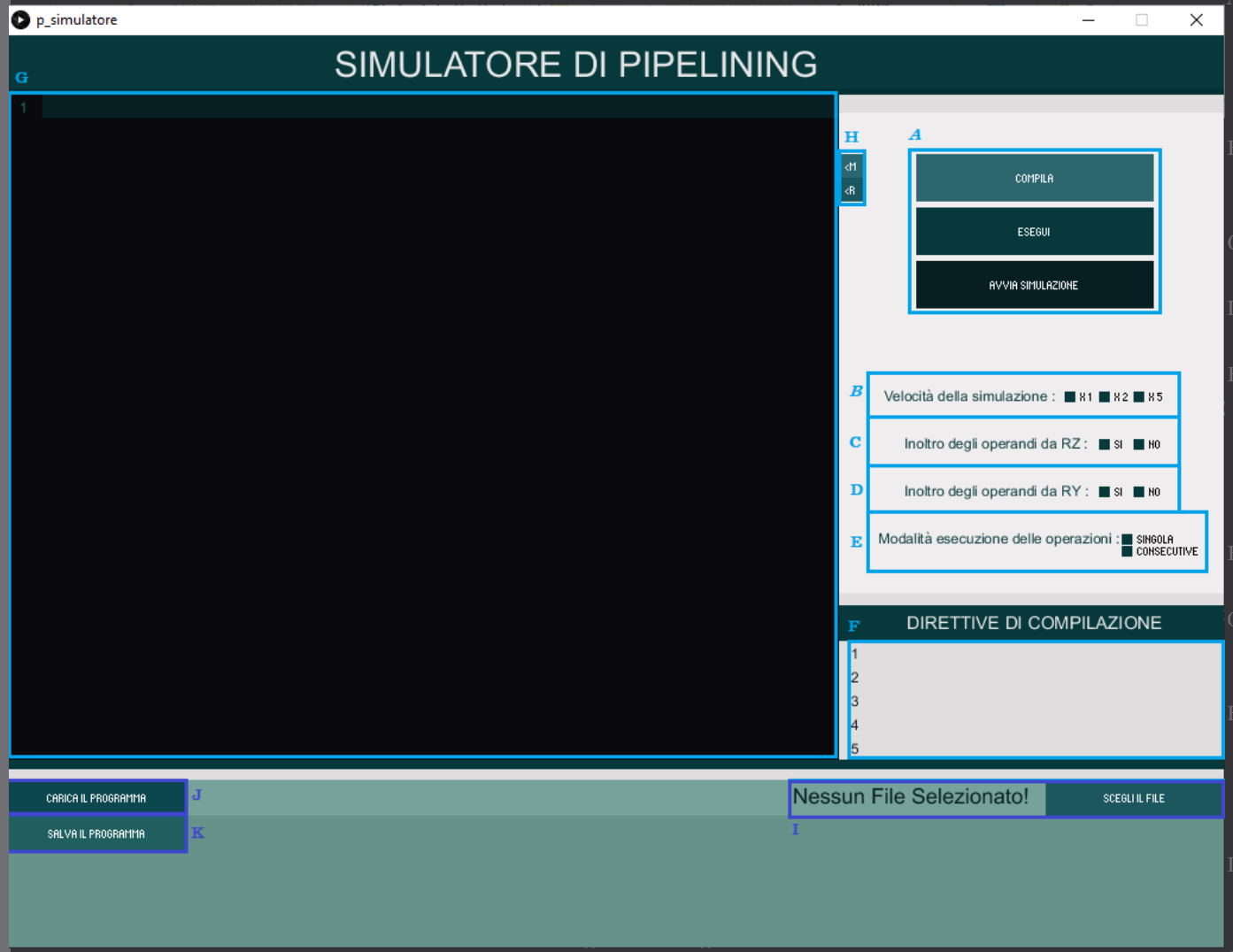
La tecnica di pipelining, sebbene aumenti sensibilmente il troughtput del processore, non è priva di difetti. L'esecuzione concorrente di più istruzioni può portare a degli stalli provocati da:

-Dipendenza tra dati > Si verifica quando il registro o indirizzo destinazione di un'istruzione è utilizzato da un'altra istruzione come sorgente quando la scrittura non è ancora avvenuta. Dovrà quindi attendere il termine dell'operazione precedente creando una «bolla», cioè uno o più cicli di clock di inattività. Il costo della dipendenza può essere mitigato attraverso l'uso della tecnica di inoltro degli operandi.

-Ritardo della memoria> Si verifica in caso di cache miss.

-Ritardo nei salti> Si verifica quando un'istruzione di salto viene prelevata. Prima di conoscere l'indirizzo della prossima istruzione da prelevare, altre istruzioni potrebbero essere prelevate dal processore. Nel caso il salto avvenga, le istruzioni prelevate dovranno essere annullate. Il costo del ritardo di salto può essere mitigato occupando il posto di ritardo di salto o predicendo il risultato del confronto nel salto condizionato.

Layout della schermata principale



A. Bottone «*COMPILA*» : compila il programma.
Bottone «*ESEGUI*» : esegue il programma se la compilazione è andata a buon fine. Bottone «*AVVIA SIMULAZIONE*» : passa alla schermata di simulazione.

B. Permette di scegliere la velocità con la quale un'operazione viene eseguita e si muove nel grafico della simulazione (**DEFAULT= x1**).

C. Attiva l'inoltro degli operandi dal registro RZ (**DEFAULT=NO**).

D. Attiva l'inoltro degli operandi dal registro RY (**DEFAULT=NO**).

E. Permette di scegliere tra due modalità di esecuzione delle istruzioni:
«*SINGOLA*» : Un ciclo di clock alla volta. L'esecuzione è controllata dall'utente (**DEFAULT**).
«*CONSECUTIVE*» : Tutto d'un fiato! L'esecuzione è controllata dal programma.

F. Parte riservata all'inserimento di direttive di compilazione(**MAX 5**).

G. (*area in alto a sinistra*) Parte riservata all'inserimento di istruzioni (**MAX 28**).

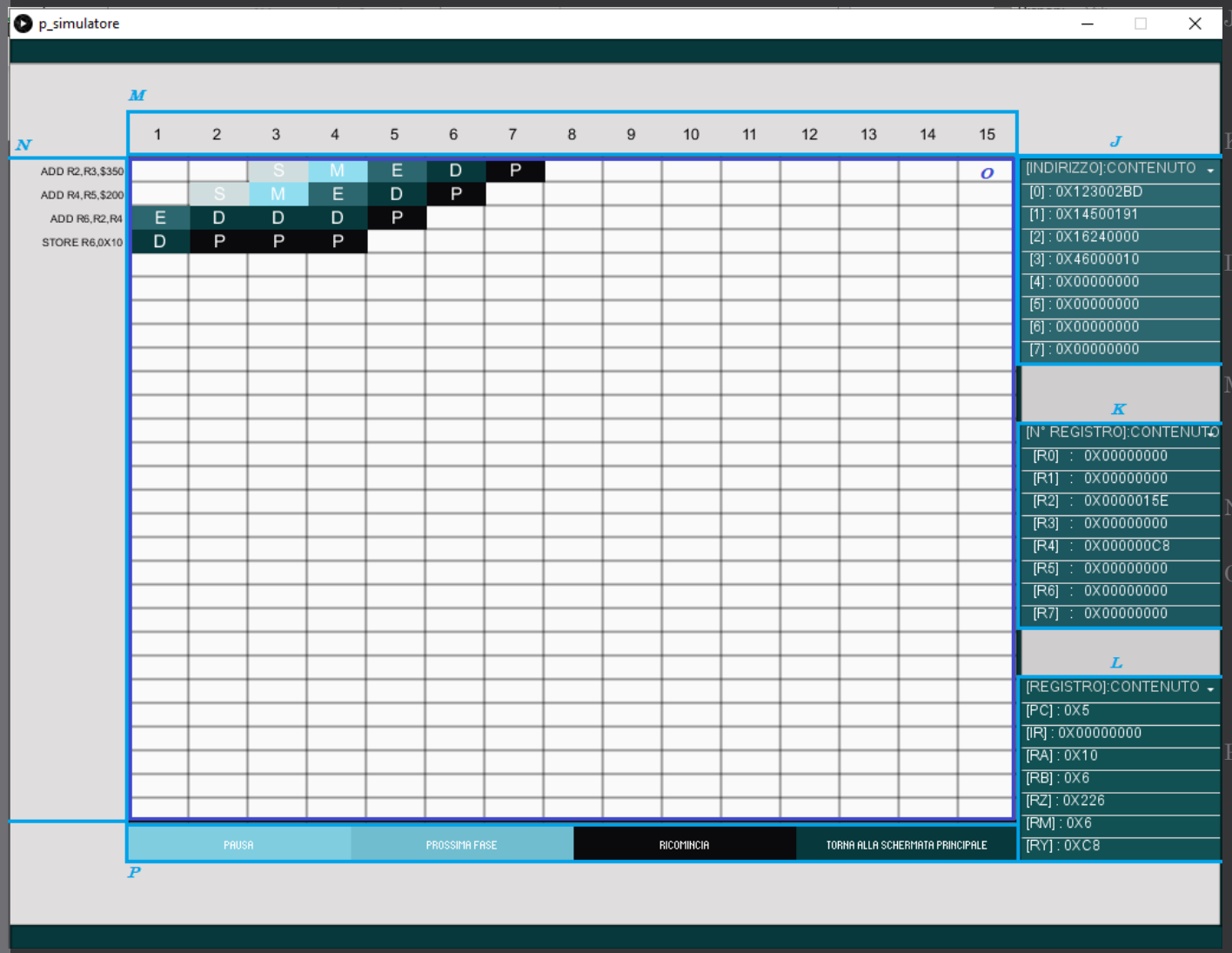
H. Bottone «<M» : permette di mostrare o nascondere il contenuto della memoria.
Bottone «<R» : permette di mostrare o nascondere il contenuto dei registri d'uso generale.

I. Il bottone «SCEGLI IL FILE» apre il selettore file. Alla sua sinistra viene mostrato, quando selezionato, il nome del file.

J. Permette di caricare il programma contenuto nel file selezionato.

K. Permette di salvare il programma scritto sotto forma di file di testo(.txt).

Layout della schermata di simulazione



J. Lista scorrevole contenente , per ogni indirizzo di memoria(80 in totale), il contenuto in formato esadecimale. Viene aggiornata a runtime del programma.

K. Lista scorrevole contenente, per ogni registro di uso generale(10 in totale) il contenuto in formato esadecimale. Viene aggiornata a runtime del programma.

L. Lista contenente il valore attuale del PC, dell'IR e dei registri interstadi RA,RB,RZ,RM e RY in formato esadecimale. Viene aggiornata a runtime del programma.

M. Ascissa del grafico. Indica l'n-esimo ciclo di clock precedente al ciclo di clock attuale(ES: Una fase situata nella sesta colonna del grafico è stata eseguita 6 cicli di clock fa).

N. Ordinata del grafico. Indica l'n-esima istruzione del programma.

O. Grafico della simulazione. Elemento fondamentale che mostra l'esecuzione del programma nel tempo. Ogni istruzione è suddivisa in 5 fasi che vengono rappresentate graficamente con un rettangolo colorato ed una sigla (P: prelievo , D: decodifica, E:esecuzione , M: accesso in memoria, S: scrittura);

P. Bottone «*PAUSA*» : se in esecuzione, mette in pausa. Se in pausa, riprende l'esecuzione.
Bottone «*PROSSIMA FASE*» : attivo solo in modalità di esecuzione «*SINGOLA*». Passa al ciclo di clock successivo.
Bottone «*RICOMINCIA*» : ricomincia la simulazione del programma.
Bottone «*TORNA ALLA SCHERMATA PRINCIPALE*» : torna alla schermata principale.

Modalità di utilizzo

- **Scrittura del programma.**

Per cominciare a scrivere un programma basta cliccare sulla prima riga della zona riservata alla scrittura di istruzioni. Essa verrà evidenziata e un cursore apparirà al suo interno, permettendo la scrittura. Premendo il tasto ENTER o la freccia direzionale verso il basso una nuova istruzione verrà aggiunta, fino ad un massimo di 28.E' possibile navigare tra le istruzioni utilizzando le frecce direzionali verso il basso e verso l'alto. Lo stesso vale per le direttive di compilazione, ma il numero massimo di direttive è 5. Ogni istruzione deve essere scritta rispettando la sintassi qui definita, inclusi gli spazi: [***x***] => ***x*** OPZIONALE; { ***a , b , c***} => scelta OBBLIGATORIA tra ***a , b*** o ***c***

Instruzioni macchina:

[*nome_etichetta*:] add r{*0,1..9*},r{*0,1..9*},**{** r{*0,1..9*},${*0,1,..524.287*} **}** *EX*: add r1,r3,r5 => r1=r3+r5 ; add r1,r3,$10 => r1=r3+10

[*nome_etichetta*:] subtract r{*0,1..9*},r{*0,1..9*},**{** r{*0,1..9*},${*0,1..524.287*} **}** *EX*: subtract r1,r3,r5 => r1=r3-r5 ; subtract r1,r3,$10 => r1=r3-10

[*nome_etichetta*:] load r{*0,1..9*},**{** {*0x1C,0x1D..0x4F*},{nome_etichetta} **}** *EX*: load r1,0x30 =>carica valore contenuto all'indirizzo di memoria 0x30 in r1.
load r1,LBL =>carica valore contenuto all'indirizzo puntato da LBL in r1.

[*nome_etichetta*:] store r{*0,1..9*},**{** {*0x1C,0x1D..0x4F*},{nome_etichetta} **}** *EX*: store r2,LBL =>carica valore contenuto in r2 all'indirizzo puntato da LBL
store r2,0x4A => carica valore contenuto in r2 all'indirizzo di memoria 0x4A

[*nome_etichetta*:] move r{*0,1..9*},**{**r{*0,1..9*},${*0,1..8.388.607*} **}** *EX*: move r3,r4 => r3=r4 ; move r3,$200 => r3=200

[*nome_etichetta*:] branch r{*0,1..9*},r{*0,1..9*},{=,!,>,<},nome_etichetta *EX*: branch r3,r6,!,LBL => controlla che il contenuto di r3 sia diverso dal contenuto di r6.Se vero, sposta l'esecuzione all'istruzione con etichetta LBL.

Direttive di compilazione:

origin {*0x1C,0x1D..0x4F*) //Fissa l'indirizzo di memoria di partenza per il salvataggio di valori (dataword) o la riservazione di spazio.

[*nome_etichetta*:] dataword *${-2.147.483.648,..0,..2.147.483.647}* //Il valore viene salvato in memoria. L'indirizzo dipende da origin e reserve.

[*nome_etichetta*:] reserve {*1,..N*} //Riserva spazio in memoria. N è il valore soglia uguale al complemento dell'indirizzo attualmente puntato alla dimensione totale della memoria.
EX: Se l'indirizzo puntato è 1C, la soglia N sarà uguale a 0x50-0x1C = 80-28 = 52.
NOTA! La memoria è indirizzabile per parola.

Modalità di utilizzo

-Compilazione, esecuzione ed avvio della simulazione.

Dopo aver scritto il proprio programma, è necessario compilarlo usando il bottone «COMPILA». Nel caso in cui un errore venga trovato dal compilatore, un messaggio di errore verrà mostrato al disotto del pulsante «AVVIA SIMULAZIONE», altrimenti il messaggio «Programma compilato con successo!» verrà mostrato. Un errore di compilazione potrebbe consistere in un errore nella sintassi dell'istruzione o il riferimento ad un'etichetta non esistente.

E' possibile usare il bottone «ESEGUI» per eseguire immediatamente il programma e analizzare le modifiche che esso apporta alla memoria e ai registri di uso generale.

E' possibile usare il bottone «AVVIA SIMULAZIONE» sia prima che dopo l'eventuale esecuzione del programma. Nel caso nessuna delle opzioni di simulazione venga modificata, essa verrà inizializzata con le opzioni di default(*velocità x1, nessun inoltro di operandi da RZ o RY, esecuzione per singole istruzioni*).Inoltre, la pressione del bottone provocherà la transizione del programma dalla schermata principale alla schermata di simulazione.

Nel caso si volesse riavviare la simulazione dopo essere tornati alla schermata principale, sarà necessario ricompilare il programma.

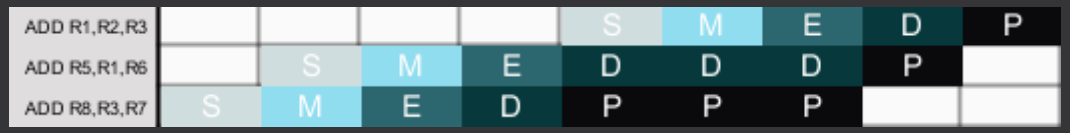
Modalità di utilizzo

-Controllo e interpretazione del grafico

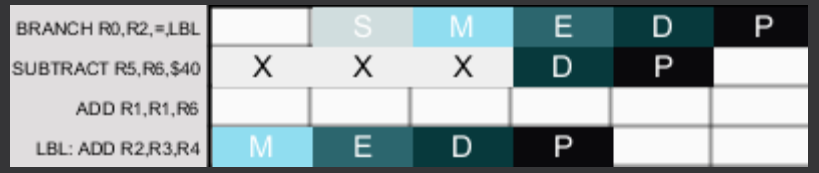
L'esecuzione del grafico può essere gestita dall'utente tramite un gruppo di pulsanti il cui funzionamento è spiegato nella pagina riguardante il layout della schermata di simulazione. Nel caso la modalità di esecuzione scelta sia «SINGOLA» dovrà essere l'utente a dire al programma quando cominciare la simulazione e quando muoversi al ciclo di clock successivo. Diversamente , in modalità di esecuzione «CONSECUTIVE» sarà il programma a cominciare la simulazione e l'utente potrà solo decidere se mettere in pausa o ricominciare la simulazione, perdendo il controllo sull'esecuzione step-by-step.

Ogni riga del grafico rappresenta un'istruzione. Ogni istruzione deve passare attraverso 5 fasi per essere completata. La durata di ogni fase dipende dal rapporto tra le istruzioni del programma. Qui vengono presentati 3 esempi:

Caso standard: Nessuna dipendenza da istruzione e nessun salto. Ogni fase dura un solo ciclo di clock.



Dipendenza di dato: La seconda istruzione legge un registro che deve essere scritto da un operazione precedente. E' necessario attendere il termine della scrittura prima di usarlo



Ritardo di salto. L'operazione successiva al branch comincia l'esecuzione senza conoscere l'esito della condizione di salto. Se l'esito è positivo, l'istruzione viene annullata .
Nota: La valutazione della condizione di salto viene effettuata in fase di decodifica . Di conseguenza, il ritardo di salto sarà di un solo ciclo di clock.

Il simulatore offre la possibilità di attivare l'inoltro degli operandi da RZ e/o da RY ad uno degli ingressi dell'ALU, permettendo così di alleviare gli stalli dovuti a dipendenze di dato. Nell'esempio di dipendenza mostrato , la bolla di 3 cicli di clock non avrebbe più necessità di essere creata e l'esecuzione continuerebbe normalmente.

Modalità di utilizzo

-Salvataggio e caricamento di programmi

Dalla versione 1.1 è possibile salvare e caricare i programmi scritti con l'applicazione .

Per salvare un programma :

- Cliccare sul bottone «Salva il programma».
- Scegliere il percorso ed il nome del file.

Il file verrà salvato nel formato «txt».

Per caricare un programma :

- Cliccare sul bottone «Scegli il file» e selezionare il file contenente il programma da caricare tramite il selettore di file.
- Cliccare sul bottone «Carica il programma».

Il file caricato dovrà avere l'estensione «.txt» per essere compatibile. Un file di qualunque altro tipo sarà illeggibile per il software.

La cartella «CodeExamples» all'interno della cartella «doc» della distribuzione contiene alcuni file con esempi di codice pronti per essere caricati, molto utili per familiarizzare con il funzionamento del simulatore.

Codifica binaria delle istruzioni

Le istruzioni vengono salvate in memoria a partire dall'indirizzo 0x00 e vengono così codificate:

- Bit 31-28 : codice operativo { 0000 = NOP ; 0001 = ADD ; 0010 = SUBTRACT ; 0011 = LOAD ; 0100 = STORE ; 0101 = MOVE ; 0110 = BRANCH }

- ADD O SUBTRACT
  - Se somma o differenza tra registri : Bit 27-24 = R_dest(da 0000 a 1001) ; Bit 23-20=R_src1(da 0000 a 1001) ; Bit 19-16 =R_src2 (da 0000 a 1001) ; Bit 0 = '0' .
  - Se somma o differenza tra registro e valore immediato : Bit 27-24 =R_dest(da 0000 a 1001) ; Bit 23-20 =R_src1 (da 0000 a 1001) ; Bit 19-1 : valore immediato ; Bit 0 = '1' .

- LOAD : Bit 27-24 = R_dest(da 0000 a 1001) ; Bit 23-0 = indirizzo di memoria

- STORE : Bit 27-24 = R_src(da 0000 a 1001) ; Bit 23-0 = indirizzo di memoria

- MOVE : Bit 27-24 = R_dest(da 0000 a 1001) ; { Bit 23-20 = R_src(da 0000 a 1001) e Bit 0='0' , Bit 23-1 = Valore immediato e Bit 0='1' }

- BRANCH : Bit 27-24 e Bit 23-20 = R_src1 e R_src2 (da 0000 a 1001) ; Bit 19-18 = segno di comparazione(=,!,>,<) ; Bit 17-0 = indirizzo di memoria

NOTA: *Alcuni valori di bit sono stati lasciati inutilizzati nel caso di eventuali future aggiunte di istruzioni , registri o ampliamenti della memoria.*

RICONOSCIMENTI

Il progetto è stato creato e sviluppato da Giovanni Piccinini , studente dell'Università degli Studi di Catania , con l'auspicio che esso possa essere utile per una più immediata comprensione del concetto di pipeline di un processore.

Il progetto è totalmente aperto a critiche o rettifiche. Il recapito mail è disponibile all'indirizzo

https://www.dmi.unict.it/archelab/projects/simulatorepipelining/

Qui termina la guida utente del programma. Se sei arrivato/a fin qui , **grazie!**