Note di rilascio del simulatore di pipelining, versione 1.1

Lo sviluppo del progetto è cominciato prefissandosi un semplice obiettivo: la creazione di un software capace di , dato un programma scritto in linguaggio macchina , costruire un grafico dinamico che mostrasse il succedersi dell'esecuzione delle diverse fasi delle istruzioni del programma nel tempo. Ciò ha reso necessaria l'implementazione di alcune classi capaci di simulare il comportamento di diverse parti del calcolatore : una piccola memoria, un banco di registri di uso generale ,un assemblatore in due fasi e un processore scalare dotato di pipeline a cinque stadi.

Documentazione del codice

Per chiunque fosse interessato a studiare più approfonditamente il codice sorgente del software, è possibile reperirne la documentazione generata con NaturalDocs qui.

NOTA: Nella documentazione è presente una piccola incongruenza nel menu Files, dove i file sorgente vengono mostrati con estensione .java e non .pde , formato originale del codice sorgente . Essa è la conseguenza della soluzione triviale trovata per ovviare al problema dell'incompatibilità fra NaturalDocs e Processing. Il formato dei file è stato modificato unicamente in questo contesto e ciò non ha nessuna ripercussione sul software originale.

Architettura del software

Risulta più conveniente , ai fini di una migliore comprensione , attuare una prima suddivisione generale del software in 3 sezioni:

1) *Sezione dedicata all'interfaccia utente*

2) *Sezione dedicata alla parte grafico/simulativa del software*

3) *Sezione dedicata alla parte logico/operativa del software*

1) Tutti i metodi relativi a tale sezione si trovano all'interno della scheda "p_simulatore". L'interfaccia è stata sviluppata utilizzando "controlP5", una libreria di Processing che implementa il design pattern Front Controller. Ogni oggetto dell'interfaccia (bottoni , liste, caselle di spunta, zone di testo) è collegato ad un Handler che viene notificato ogni volta che un controllore invia una notifica di evento avvenuto(pressione di un bottone, spunta di una casella, ...).Sarà l'Handler a riconoscere il mittente, richiamando la routine ad esso corrispondente. Hanno una particolare importanza i bottoni di caricamento e salvataggio di programmi. La codifica di un file di salvataggio è estremamente semplice: Tutte le istruzioni vengono salvate così come sono scritte in un file di testo. La stringa "direttive:" separa la parte contenente le istruzioni in assembly dalle direttive di compilatore.

2) Le classi più strettamente legate a tale sezione sono "Grafico" e "Stato".

- La classe "Grafico" si occupa della costruzione del grafico che mostrerà , per ogni istruzione , il suo progresso all'interno della pipeline del processore in rapporto col tempo di esecuzione.

- La classe "Stato" implementa il concetto di stato(o fase) all'interno del grafico. Dato in input un numero che indica la fase, la posizione assoluta e la velocità di movimento, essa disegnerà un rettangolo il cui colore

e sigla d'identificazione dipende dalla fase e che si muoverà dal limite sinistro al limite destro del grafico, fino a sparire.

È importante notare che tutti gli elementi geometrici , di testo e interattivi visualizzabili nel programma vengono generati e mantenuti visibili dalla funzione "draw" , fulcro di Processing. Essa viene eseguita ciclicamente(60 volte al secondo di default) fino al termine dell'applicazione.

3) Le classi più strettamente legate a tale sezione sono "BufferInterstadio" , "Compilatore" "Memoria", "Registro" e "ProcessoreP".

-La classe "BufferInterstadio" implementa il concetto di buffer interstadio della pipeline. Il processore ne utilizza quattro per mantenere le informazioni utili ad un'istruzione che procede all'interno della pipeline.

-La classe "Compilatore" implementa il concetto di compilatore a due fasi. Al suo interno si trovano tutti i metodi che si occupano di decodificare un'istruzione scritta in linguaggio macchina, rilevare e notificare l'eventuale presenza di errori e salvare l'istruzione in memoria sotto forma di codice binario. Si occupa inoltre di costruire la tabella delle etichette.

-La classe "Memoria" implementa il concetto di memoria del calcolatore. Al suo interno si trovano tutti i metodi che si occupano di prelievo/scrittura di dati da/in memoria.

-La classe "Registro" implementa il concetto di registro del calcolatore. Al suo interno si trovano tutti i metodi che permettono la lettura o scrittura del registro.

-La classe "ProcessoreP" è la più importante fra tutte. Implementa il concetto di processore scalare dotato di pipeline. Mantiene i riferimenti alla memoria e al banco di registri. È possibile suddividere l'ensemble dei suoi metodi in più gruppi con funzioni diverse :

Funzioni per l'esecuzione di una specifica fase	Sono 5. Ogni funzione simula ciò che avverrebbe all'interno di un processore in una delle 5 fasi d'esecuzione di un'istruzione.
Funzioni per il salvataggio di informazioni nei buffer	Sono 4 , come il numero dei buffer. Ogni funzione viene chiamata al termine di una fase (tutte esclusa la fase di write back) e permette il salvataggio di dati utili all'istruzione all'interno di un buffer interstadio.
Funzioni per la simulazione di pipelining	Si occupano di simulare il flusso di istruzioni all'interno della pipeline. Include metodi che si occupano di rilevare la presenza di dipendenze di dato, creando una bolla di attesa, e metodi che permettono l'eventuale inoltro di operandi a ciò che nel software rappresenta un'approssimazione funzionale dell'ALU.
Funzioni per l'inoltro di dati all'interfaccia utente	Si occupano di inviare dati (contenuto della memoria, contenuto dei registri di uso generale, contenuto dei registri interstadio) ai controllori dell'interfaccia utente, che ne permetteranno la visualizzazione.
Funzioni ausiliarie	Si occupano di svolgere compiti utili all'oggetto come : reset del processore, reset dei registri interstadio , reset del banco di registri.

Interpretazione dei valori contenuti nei registri interstadio

Il simulatore permette la visualizzazione dinamica dei contenuti della memoria, dei registri di uso generale e dei registri interstadio. Per i primi due si rimanda alla guida utente. Verrà qui chiarito in che modo i registri interstadio vengono utilizzati dalle istruzioni, così da poter comprendere il rapporto tra il loro contenuto e le istruzioni in esecuzione.

Risulta inoltre necessario fare alcune premesse relative al funzionamento dell'applicazione, al fine da eliminare alcune ambiguità che potrebbero risultare in conseguenze più o meno gravi.

- Durante l'esecuzione della simulazione vengono mostrati i contenuti sia dei registri interstadio che dei registri di uso generale. Il loro valore viene aggiornato al termine dell'esecuzione di tutte le fasi del ciclo di clock corrente. Conseguentemente, i valori mostrati all'utente rispecchieranno i loro valori nel successivo ciclo di clock. Ciò deve essere tenuto in considerazione quando si affrontano esercizi che richiedono di indicare il valore dei registri durante le fasi d'esecuzione di una certa istruzione.

- La decisione di salto delle operazioni BRANCH viene presa durante la fase di decodifica e non durante la fase di elaborazione. Ciò significa che, in caso di salto, la penalità di salto non sarà di due cicli, ma di uno.

- L'uso dei registri interstadio è leggermente diverso da quello specificato nel testo Hamacker et al. dove: RA ed RB contengono il valore letto dai registri generali e non il loro indice, RB contiene sempre il valore del registro generale con indice contenuto in un campo fisso della codifica binaria dell'istruzione e il valore di RM viene sempre passato da RB.

-Istruzioni **ADD e SUBTRACT**:

Fase 1: Il codice dell'istruzione è salvato nel registro IR.

Fase 2: Se la somma/sottrazione avviene tra due registri, in RA ed RB verranno salvati rispettivamente gli indici del primo e del secondo registro sorgente.
Se la somma/sottrazione avviene tra registro e valore immediato, in RA verrà salvato l'indice del primo registro sorgente mentre il valore di RB dipenderà dalle operazioni precedenti.

Fase 3: Se la somma/sottrazione avviene tra due registri , in RZ verrà salvato il risultato dell'operazione che ha per operandi il contenuto del registro puntato da RA ed il contenuto del registro puntato da RB.
Se la somma/sottrazione avviene tra registro e valore immediato, in RZ verrà salvato il risultato dell'operazione che ha per operandi il contenuto del registro puntato da RA e il valore immediato.
In RM verrà salvato l'indice del registro destinazione.

Fase 4: Il valore di RZ viene passato a RY.

Fase 5: Il valore di RY viene salvato nel registro di indice RM.

-Istruzione **LOAD**

Fase 1: Il codice dell'istruzione è salvato nel registro IR.

Fase 2: L'indirizzo di memoria viene salvato in RA, l'indice del registro destinazione viene salvato in RB.

Fase 3: Il valore di RA viene passato a RZ, il valore di RB viene passato ad RM.

Fase 4: Il valore puntato dall'indirizzo contenuto in RA viene salvato in RY.

Fase 5: Il valore di RY viene caricato nel registro di indice RM.

-Istruzione **STORE:**

Fase 1: Il codice dell'istruzione è salvato nel registro IR.

Fase 2: L'indirizzo di memoria viene salvato in RA, l'indice del registro sorgente viene salvato in RB.

Fase 3: Il valore di RA viene passato a RZ, il valore di RB viene passato a RM.

Fase 4: Il valore del registro di indice RM viene salvato all'indirizzo contenuto in RZ.
Il valore di RY dipende dalle operazioni precedenti.

Fase 5: Nessuna operazione.

-Istruzione **MOVE**:

Fase 1: Il codice dell'istruzione è salvato nel registro IR.

Fase 2: Se l'elemento da muovere è un registro, in RA ed RB verranno salvati rispettivamente l'indice del registro destinazione e l'indice del registro sorgente.
Se l'elemento da muovere è un valore immediato , in RA verrà salvato l'indice del registro destinazione mentre il valore di RB dipenderà dalle operazioni precedenti.

Fase 3: Se l'elemento da muovere è un registro, il valore di RA verrà salvato a RZ ed il valore di RB verrà salvato ad RM.
Se l'elemento da muovere è un valore immediato , il valore di RA verrà salvato in RZ ed il valore immediato verrà salvato in RM.

Fase 4: Il valore di RZ viene salvato in RY.

Fase 5: Il valore del registro di indice RM viene salvato nel registro di indice RY.

-Istruzione **BRANCH:**

Fase 1: Il codice dell'istruzione è salvato nel registro IR.

Fase 2: In RA ed RB vengono salvati gli indici dei due registri di destinazione. I contenuti dei registri di indice RA ed RB vengono confrontati. Se la condizione di salto è rispettata , l'esecuzione salta all'indirizzo di destinazione.

Fase 3:Il valore di RZ ed RM dipende dalle operazioni precedenti.

Fase 4: Nessuna operazione. Il valore di RY dipende dalle operazioni precedenti.

Fase 5: Nessuna operazione.

Tecniche di programmazione

Il progetto è stato sviluppato interamente su Processing, un'IDE che implementa il linguaggio di programmazione ad oggetti JAVA , aggiungendo varie funzioni che semplificano la creazione di applicazioni multimediali interattive.

Idee per ulteriori sviluppi

Al momento il software presenta varie limitazioni alle quali sarebbe utile ovviare con:

-Aumento del numero massimo di istruzioni , correntemente fissato a 28

-Aumento del numero massimo di direttive , correntemente fissato a 5

-Possibilità di controllare dinamicamente la dimensione della memoria

-Aggiunta di nuove istruzioni all'IS(Instruction Set) del software

~~-Aggiunta della possibilità di salvare o caricare programmi su/da file di testo~~ ☑

-Aggiunta di possibilità di predire se un salto avverrà o no , con automa a 2 o a 4 stati

-Aggiunta di possibilità di occupare la posizione di ritardo di salto

-Miglioramento del grafico degli stati delle istruzioni

-Refactoring del codice

-aggiunta di funzionalità di editing del programma (e.g. inserimento e cancellazione di istruzioni e direttive).