ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

Dipartimento di Informatica - Scienza e Ingegneria

Corso di Laurea Triennale in Ingegneria Informatica

TESI DI LAUREA

in

Calcolatori Elettronici - T

Estensione di un simulatore del processore DLX

CANDIDATO

Filippo Comastri

RELATORE

Stefano Mattoccia

Anno Accademico: 2020/2021

1 INTRODUZIONE

Lo scopo di questa tesi di laurea è quello di estendere e dove possibile migliorare un simulatore DLX e RISC-V già sviluppato in precedenza nelle tesi degli studenti Alessandro Foglia [1], Fabrizio Maccagnani [2] e Federico Pomponii [3].
Il simulatore funge da strumento didattico per gli studenti del corso universitario Calcolatori Elettronici-T dell'Università di Bologna che grazie ad esso possono approfondire e capire al meglio il funzionamento di un vero processore.
A tal proposito va tenuto conto che l'obiettivo del simulatore è quello di riprodurre in maniera fedele il comportamento di un processore con architettura DLX o eventualmente RISC-V ma senza replicare il funzionamento interno di un processore fisico. La tesi si focalizzerà sulla simulazione del processore DLX.

Parte del lavoro ha riguardato anche il deployment del simulatore su un server del Dipartimento di Informatica – Scienza e Ingegneria (DISI) dell'Università di Bologna, per rendere disponibile l'applicazione a ciascuno studente dai propri dispositivi [7].

1.1 Architettura RISC-V

Il RISC-V è uno standard open-source di istruzioni ISA (Instruction Set Architecture) di tipo RISC (Reduced Instruction Set Computer). Sebbene sia conveniente parlare dell'ISA RISC-V, RISC-V è in realtà una famiglia di ISA correlati, di cui attualmente esistono quattro ISA di base [1].
Esistono due varianti per i set di istruzione base degli interi, RV32I e RV64I, che forniscono rispettivamente spazi di indirizzi a 32 o 64 bit. La versione presa in esame è RV32I, utile anche a scopo didattico, e sarà utilizzato il termine XLEN per fare riferimento alla larghezza in bit di un registro [1].

1.2 Architettura DLX

DLX è un'architettura di microprocessori RISC ideata da John Hennessy e Dave Patterson, viene utilizzata a scopo didattico per la sua semplicità di realizzazione. L'architettura DLX come tutte le architetture RISC consiste in un insieme di istruzioni ridotto, e molti registri ad utilizzo generale detti GPR (General Purpose Registers), a differenza delle architetture CISC (Complex Instruction Set Computer), che hanno molte più istruzioni complesse e meno registri GPR [2].
L'ISA DLX possiede un unico spazio di indirizzamento di 4 Gigabyte e definisce 32 registri General Purpose (GPR) a 32 bit, R0-R31. Il registro R0 ha sempre valore 0, mentre il registro R31 è usato per salvare l'indirizzo di ritorno per le istruzioni JAL

(Jump And Link) e JALR (Jump And Link Register). I GPR possono essere usati come operandi o come destinazione della maggior parte delle istruzioni.

1.2.1 Modalità di accesso alla memoria

Il DLX prevede un'unica modalità di indirizzamento: indiretto. L'indirizzamento indiretto prevede che l'indirizzo di accesso alla memoria sia ottenuto sommando un valore costante presente nell'istruzione con il contenuto di un registro.

Indirizzo = costante + registro

Il registro è cablato nell'istruzione ma il suo contenuto può cambiare a tempo di esecuzione. Nel DLX l'indirizzo (a 32 bit) è sempre ottenuto sommando un registro a 32 bit con un valore immediato a 16 bit esteso a 32 bit con segno [4].

1.2.2 Istruzioni DLX

Le istruzioni DLX hanno lunghezza costante di 32 bit e sono presenti 3 diversi formati: I, R, J.

1.2.2.1 Tipo I

Figura 1.1

Include le istruzioni di Load e Store, operazioni ALU con immediato, le istruzioni di Branch condizionali e le istruzioni di Jump incondizionali Jump Register (JR) e Jump And Link Register (JALR). Il formato delle istruzioni di tipo I è illustrato nella Figura 1.1.
Il campo *OpCode* specifica quale istruzione DLX debba essere eseguita, ed è comune a tutti i tipi di istruzione [2].
Nelle operazioni Load e ALU *RS2/Rd* è il registro destinazione (*Rd*). Nelle Store *RS2/Rd* è il registro che contiene il valore da salvare in memoria (*RS2*).
Il campo *RS1* indica un registro che viene utilizzato come argomento dell'istruzione.

1.2.2.2 Tipo R

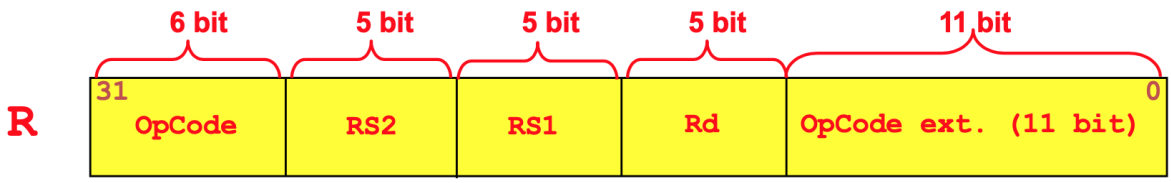

Figura 1.2

Le istruzioni di tipo R sono usate per le operazioni ALU tra registri e per copiare il contenuto di registri GPR nei registri speciali e viceversa. Le istruzioni di tipo R del DLX a numeri interi hanno un solo *OpCode* e le diverse operazioni sono distinte dal campo *OpCode ext*.
Il campo *RS1* indica il registro GPR che viene utilizzato come primo operando dell'istruzione.
Il campo *RS2* indica il registro GPR che viene utilizzato come secondo operando dell'istruzione.
Il campo *Rd* indica il registro GPR destinazione del risultato dell'operazione.
Il campo *OpCode ext.* serve a distinguere quale operazione deve eseguire la ALU.

1.2.2.3 Tipo J

Figura 1.3

Le istruzioni di tipo J includono Jump (J), Jump And Link (JAL), TRAP, e Return From Exception (RFE).
Il campo *OpCode* specifica quale istruzione DLX debba essere eseguita.
Il campo *immediato/offset* contiene l'offset che viene aggiunto all'indirizzo successivo del Program Counter (PC + 4) per ottenere il nuovo indirizzo. Per l'istruzione TRAP il campo immediato/offset rappresenta un indirizzo assoluto, mentre per l'istruzione RFE non è utilizzato.

2 STRUMENTI UTILIZZATI

2.1 Angular

Il simulatore è stato sviluppato con Angular, uno dei framework Javascript più popolari per sviluppare applicazioni web a singola pagina.
AngularJS è la prima versione del framework che fu sviluppata nel 2009 con l'intenzione di poter facilitare la realizzazione delle applicazioni attraverso l'uso di un'estensione del linguaggio HTML.
A partire da Angular 2 il framework fu completamente riscritto rispetto ad AngularJS, rendendo le successive versioni incompatibili con quest'ultima. Quando si parla di Angular ci si riferisce solitamente alle versioni dalla seconda in poi che furono rilasciate a partire dal 2014 e che ebbero un notevole successo. Ad oggi siamo alla versione 12.1.1 rilasciata a giugno 2021.
Lo sviluppo di una nuova versione di Angular ha permesso di renderlo più semplice e snello e di diminuire il tempo necessario per imparare ad utilizzare il framework.
Le applicazioni Angular sono strutturate in Componenti che facilitano la riusabilità e la manutenzione dell'applicazione.
Angular supporta Javascript ma è consigliato l'utilizzo del linguaggio Typescript, sviluppato da Microsoft, che è un'estensione di Javascript.
JavaScript è un linguaggio tipizzato, che non prevede controlli statici sui tipi di dato ed effettua una conversione implicita tra tipi (dynamic type-checking).
Questa caratteristica favorisce l'enorme flessibilità del linguaggio ed è probabilmente una delle chiavi successo di JavaScript, tuttavia l'assenza di controlli sui tipi durante la compilazione (a "compile time") può generare errori difficili da analizzare a runtime.
Per colmare questa lacuna, Microsoft ha implementato nel 2012 una estensione di JavaScript che aggiunge il supporto per il controllo statico dei tipi e altre funzionalità pensate per scrittura di applicazioni complesse quali TypeScript.
Dunque, un'applicazione TypeScript viene tradotta da un compilatore (transpiler) in un'applicazione JavaScript standard eseguibile su qualsiasi engine. La compilazione genera il codice Javascript ottimizzato e con i dovuti controlli di tipo [6].

È stato inoltre messo a disposizione degli sviluppatori uno strumento come Angular CLI (Command Line Interface) che semplifica notevolmente la fase di creazione, sviluppo, test e deployment di un progetto tramite semplici comandi.
In particolare, nel progetto è stato utilizzato il comando *ng build* che compila l'applicazione in una directory predefinita denominata */dist*. Il semplice comando *ng build*, senza altre opzioni, genera quindi un'applicazione in tempi brevi, ma non è

pronta per essere distribuita. È possibile creare una versione ottimizzata di un'applicazione utilizzando l'opzione *--prod* (production) del comando *ng build*. In questo caso Angular CLI impiega una serie di accorgimenti e ottimizzazioni:

- viene utilizzato il meccanismo di compilazione AOT (Ahead-of-Time Compilation) che converte il codice HTML e TypeScript in codice javascript efficiente durante la fase di build, dunque prima che il browser scarichi ed esegua quel codice. In questo modo l'applicazione avrà tempi di download, di parsing e di esecuzione più bassi. I file javascript avranno complessivamente una dimensione inferiore, considerato soprattutto che viene rimosso il compilatore JIT (Just-In-Time) non più necessario;
- vengono utilizzate tecniche di minificazione, uglification, tree-shaking e dead code elimination per ottimizzare al massimo il codice finale;
- viene separato il codice CSS all'interno di fogli di stile [11];

Nel progetto è stato anche utilizzato Angular Material, una libreria grafica per Angular, che aggiunge gli elementi grafici di base seguendo le specifiche di Material Design.

Per le icone è stato utilizzato Font Awesome che permette, specificando solamente un prefisso di stile e il nome dell'icona, di utilizzarle ovunque.

2.2 Ubuntu e Apache

La macchina che ospita il simulatore è una macchina virtuale Ubuntu 20.04.2 LTS appartenente al Dipartimento di Informatica – Scienza e Ingegneria (DISI) dell'Università di Bologna.
Per far girare il simulatore sul server è stato installato su quest'ultimo il server web Apache. Un web server è un applicativo software che gira su di un server fisico in ascolto su porte a lui dedicate, per comunicare con i client che richiedono i suoi contenuti ed i suoi servizi [8]. Apache è un Web Server open source utilizzato da circa il 46% dei siti in circolazione. Il nome ufficiale è Apache HTTP Server ed è sviluppato dalla Apache Software Foundation. È uno dei software più affidabili in circolazione ed esiste dal lontano 1995. Utilizzare Apache presenta diversi vantaggi essendo: open source e gratuito, affidabile e stabile, aggiornato di frequente, molto flessibile grazie alla sua struttura modulare, semplice da configurare e cross-platform [9].

2.3 Sistema di controllo delle versioni – Git

Nel corso del progetto è stato anche utilizzato un sistema di controllo delle versioni, il quale aiuta gli sviluppatori a tracciare e gestire le modifiche al codice di un progetto

software. Il controllo delle versioni permette di lavorare in sicurezza attraverso il branching (ramificazione) e il merging (fusione).
Mediante il branching, uno sviluppatore duplica parte del codice sorgente (chiamato repository). Lo sviluppatore può quindi apportare in modo sicuro modifiche a quella parte del codice senza influenzare il resto del progetto. Inoltre, è possibile recuperare vecchie versioni del software e di far coesistere simultaneamente diverse versioni.
Nello specifico è stato usato Git, un sistema di controllo versioni distribuito, il che significa che non esiste una repository centralizzata ma ogni utente mantiene una propria repository locale e la sincronizzazione avviene scambiandosi delle patch tra i vari utenti.

3 NUOVE FUNZIONALITÀ

3.1 Base di partenza

Prima del mio lavoro il simulatore comprendeva già una serie di funzionalità, realizzate da altri studenti [1][2][3] nell'ambito della loro tesi di laurea, che verranno brevemente descritte di seguito.
È presente un editor di testo nel quale poter scrivere codice assembly con la possibilità di scegliere quale set di istruzioni utilizzare: DLX o RISC-V. Tramite un tasto è possibile salvare nella memoria locale del browser il contenuto dell'editor di testo.
In una sezione apposita è disponibile la documentazione delle istruzioni DLX o RISC-V.
È possibile eseguire il codice osservando a tempo di esecuzione lo stato dei registri. Durante l'esecuzione è anche possibile generare un interrupt. Si può inoltre definire da quale tag far partire l'esecuzione e l'intervallo di tempo tra l'esecuzione di un'istruzione e la successiva.
In un'apposita area è possibile aggiungere o rimuovere spazi di memoria, configurare le loro caratteristiche e visualizzare il valore di un determinato indirizzo di memoria.
In questa stessa area è possibile modificare e personalizzare la rete di avvio e utilizzare una rete logica per l'accensione e lo spegnimento di un led.
Inoltre, si può visualizzare lo schema ai morsetti delle reti logiche e modificare gli indirizzi dei Chip-Select tramite un'apposita interfaccia.

Nell'ambito di questa tesi laurea sono state aggiunte le seguenti funzionalità:

- Possibilità di salvare in locale un file di testo contenente il codice scritto nell'editor
- Possibilità di visualizzare i valori in memoria in maniera più dettagliata con l'aggiunta di un'interfaccia grafica intuitiva per navigare tra le memorie
- Salvataggio automatico nella memoria del processore della codifica di ciascuna istruzione scritta nell'editor
- Aggiunta di un contatore tra le reti logiche che possono essere utilizzate all'interno del simulatore

3.2 Salvataggio codice in locale

Per permettere agli studenti di mantenere una copia del proprio lavoro svolto sul simulatore, è stata aggiunta la possibilità di salvare il codice contenuto nell'editor di testo su un file che viene scaricato in locale.

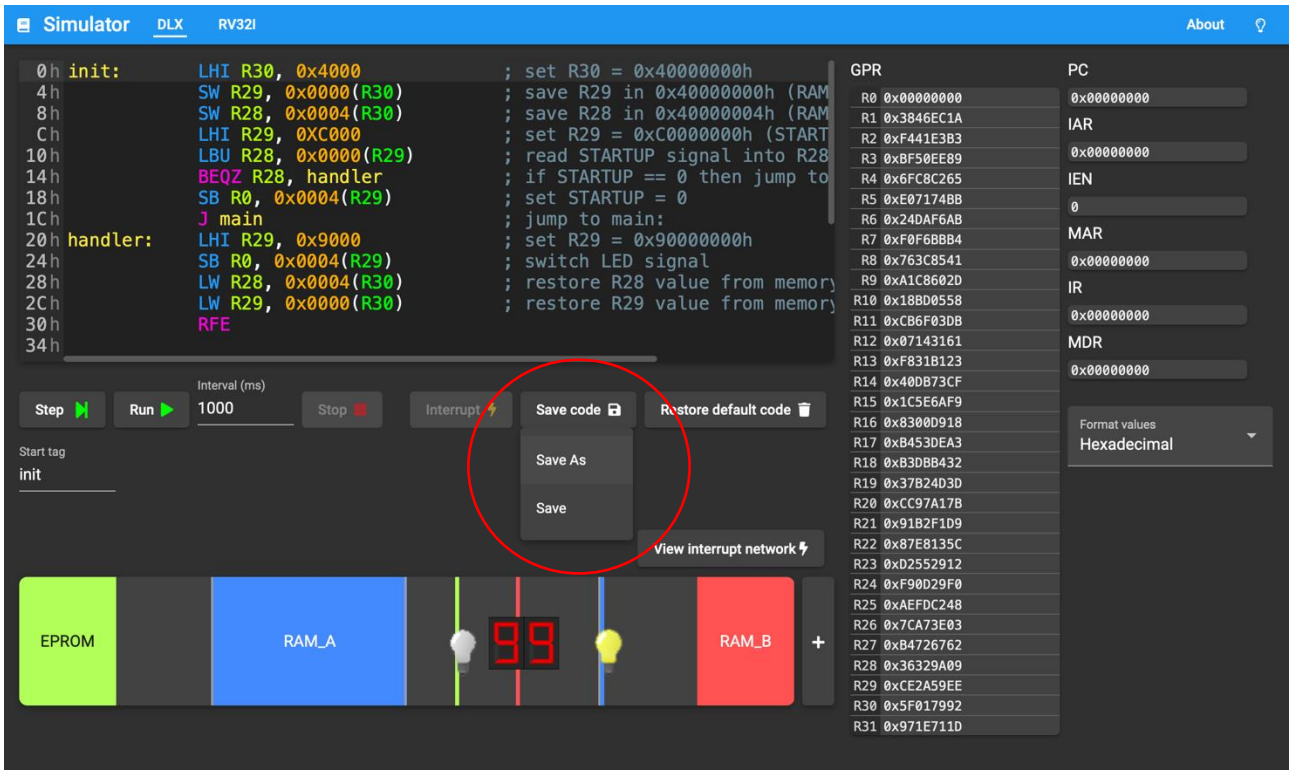

Figura 3.1

Come si vede in Figura 3.1 cliccando sul tasto "*Save Code*" si apre un menù a tendina nel quale si può scegliere che tipo di salvataggio effettuare. Cliccando su *"Save"* il contenuto dell'editor viene salvato nella memoria locale del browser. Cliccando su *"Save As"* invece si apre una finestra di dialogo (Figura 3.2) nella quale è possibile scegliere il nome del file in cui viene salvato il codice scritto nell'editor.

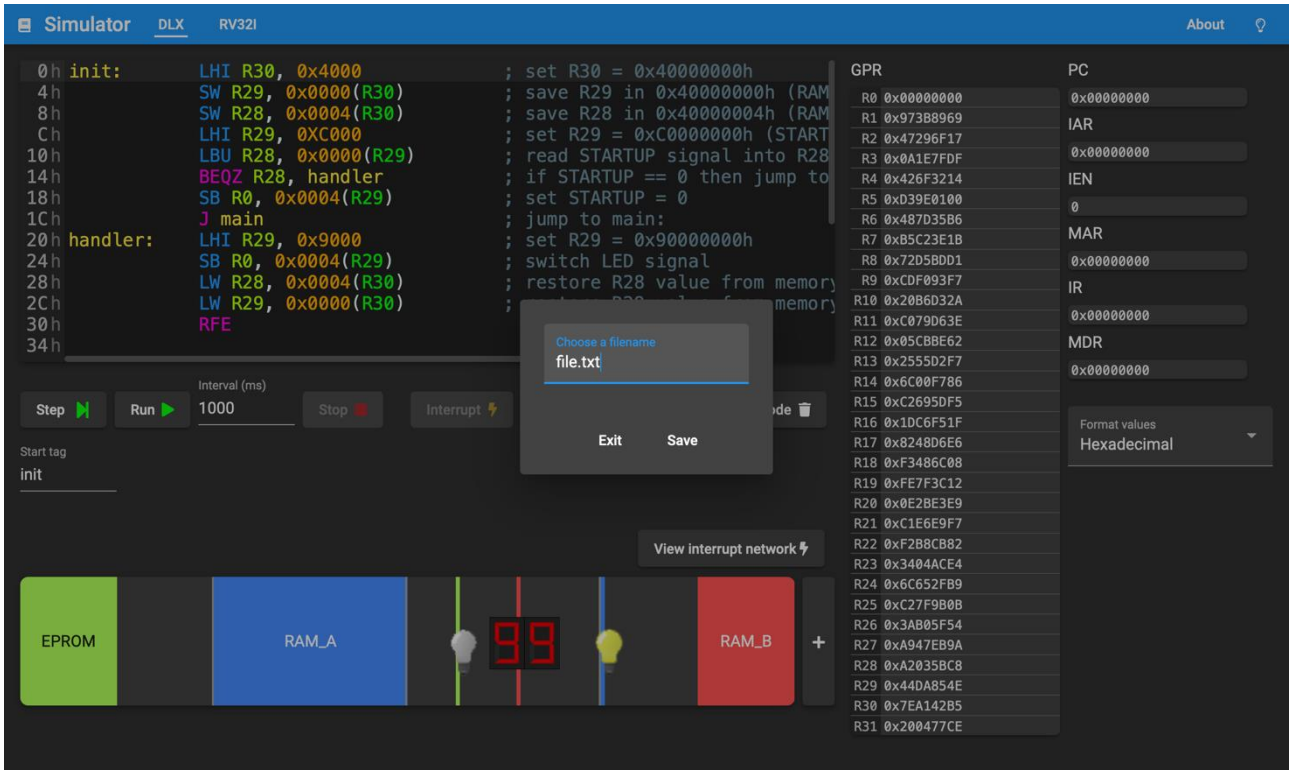

Figura 3.2

Dalla finestra di dialogo cliccando su *"Save"* il file verrà scaricato dal browser e sarà possibile visualizzarlo nella cartella dei Downloads.

3.3 Visualizzazione dettagliata della memoria

È stata apportata una miglioria alla visualizzazione dei valori in memoria, aggiungendo la possibilità di visualizzare i valori non solamente distanziati di 0004h, ma anche solamente di un byte. In questo modo è possibile analizzare in modo più preciso e puntuale il valore di ciascuna cella di memoria.

3.3.1 Visualizzazione byte per byte

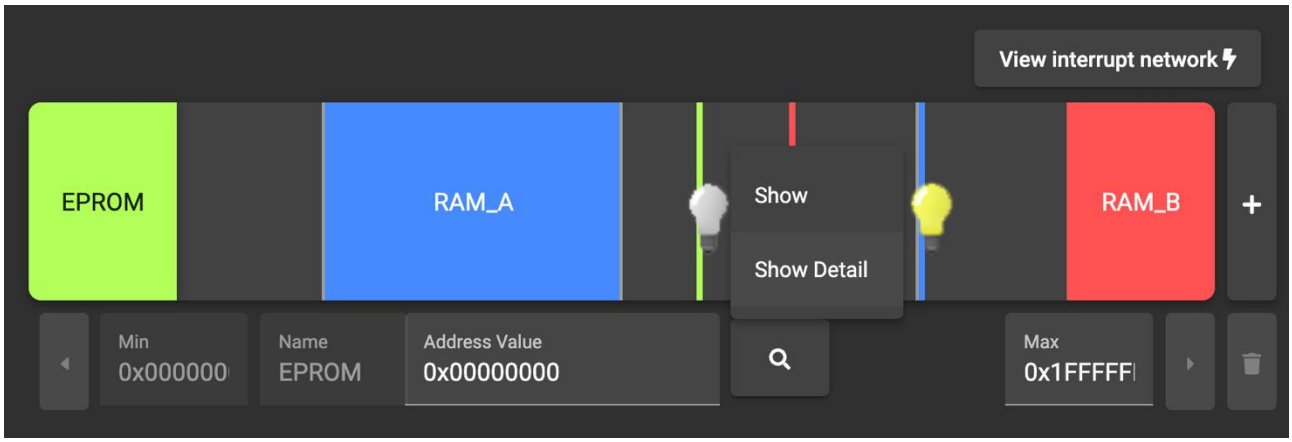

Figura 3.3

Nella sezione delle memorie (Figura 3.3), cliccando sull'icona della lente di ingrandimento, si apre un menù a tendina: cliccando su *"Show"* si visualizza il contenuto della memoria a partire dell'indirizzo scritto nel campo *"Address Value"* con gli indirizzi separati di 0004h; cliccando su *"Show Detail"* viene aperta la visualizzazione più dettagliata della memoria con gli indirizzi distanziati di un byte.

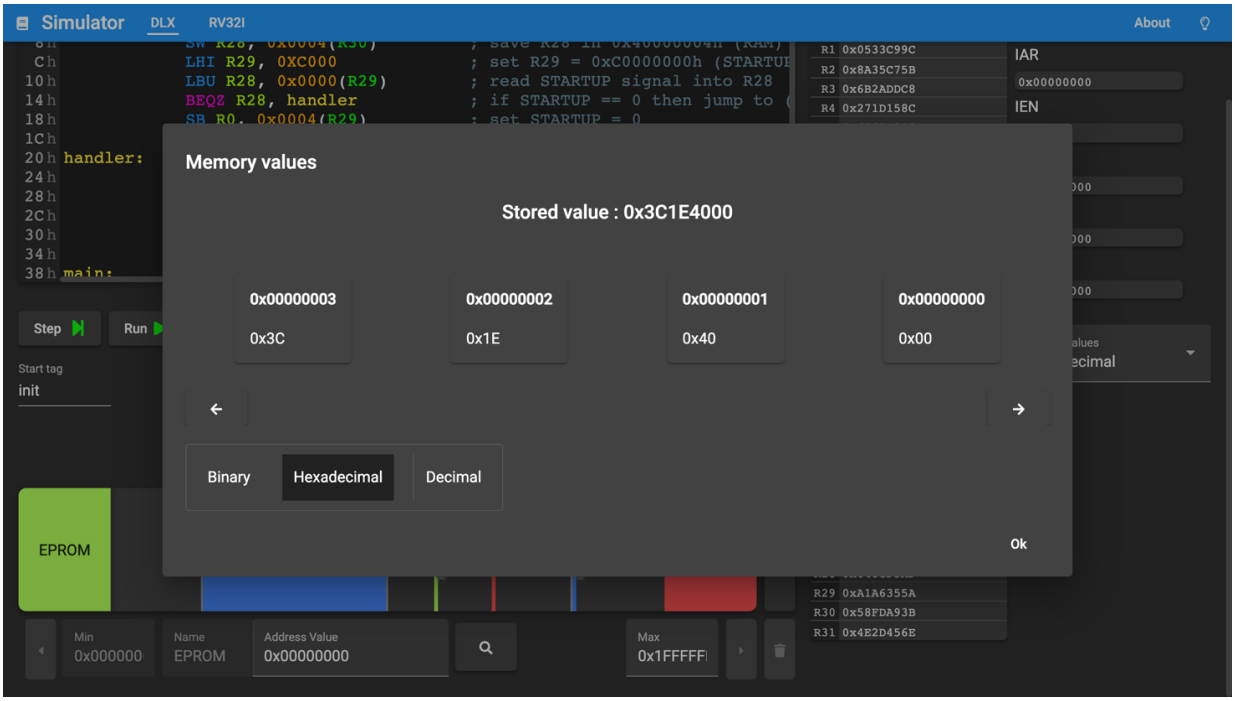

Figura 3.4

In Figura 3.4 vediamo l'interfaccia della visualizzazione dettagliata della memoria. Nei 4 blocchi vengono mostrati byte per byte i valori contenuti in memoria a partire dall'indirizzo selezionato nel campo *"Address Value"*. I valori vengono salvati in memoria rispettando la convenzione *Little Endian,* secondo la quale una grandezza numerica viene memorizzata partendo dal byte meno significativo e finendo con quello più significativo.
Come esempio osserviamo la Figura 3.4. Il valore memorizzato a partire dall'indirizzo 0x0000000 è 0x3C1E4000 (*"Stored value"*). Tale valore è lungo 4 byte, quindi ciascun byte verrà memorizzato nel range di indirizzi 0x00000000 – 0x00000003. Secondo la convenzione *Little Endian* la cella di memoria corrispondente all'indirizzo 0x0000000 conterrà il byte meno significativo (0x00), la cella corrispondente all'indirizzo 0x00000003 conterrà invece il byte più significativo (0x3C).
È anche possibile decidere in che formato visualizzare i valori salvati in memoria: esadecimale, binario o decimale.

3.3.2 Navigazione agile in memoria

Per facilitare la navigazione in memoria è stata sviluppata una funzionalità che permette di muoversi in modo intuitivo avanti e indietro tra le varie celle di memoria tramite due bottoni contenti l'icona di una freccia.

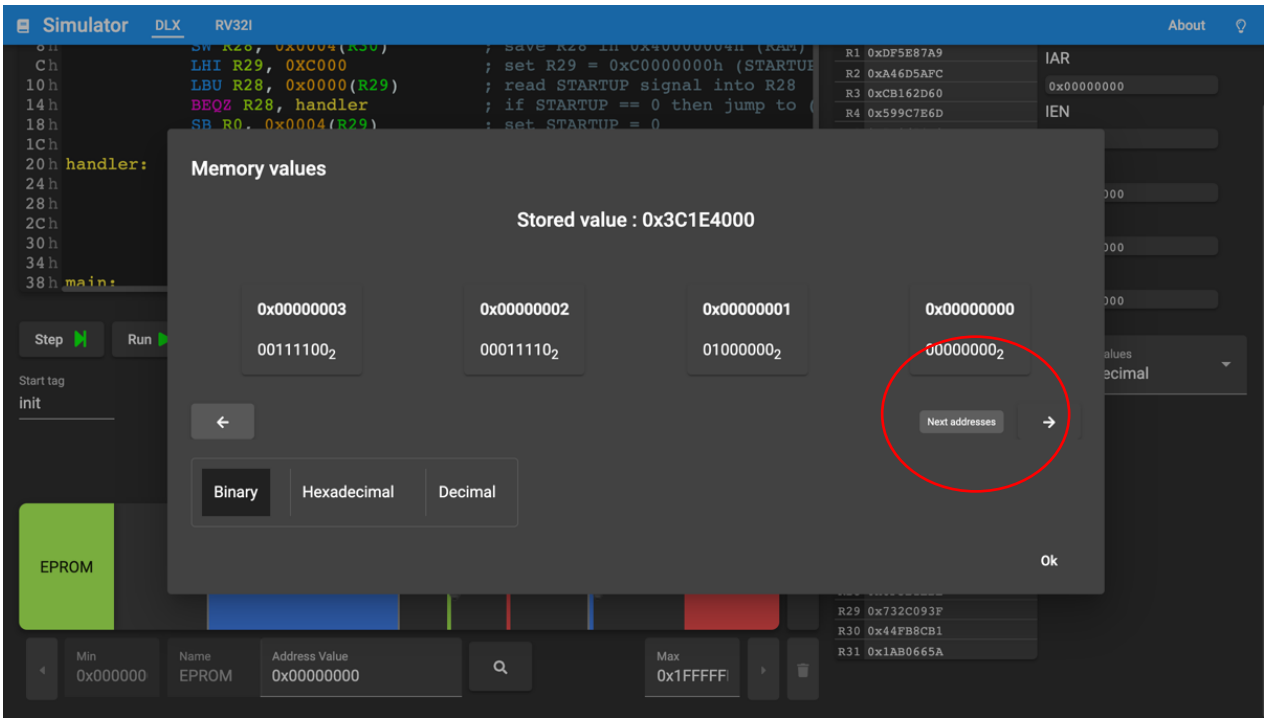

Figura 3.5

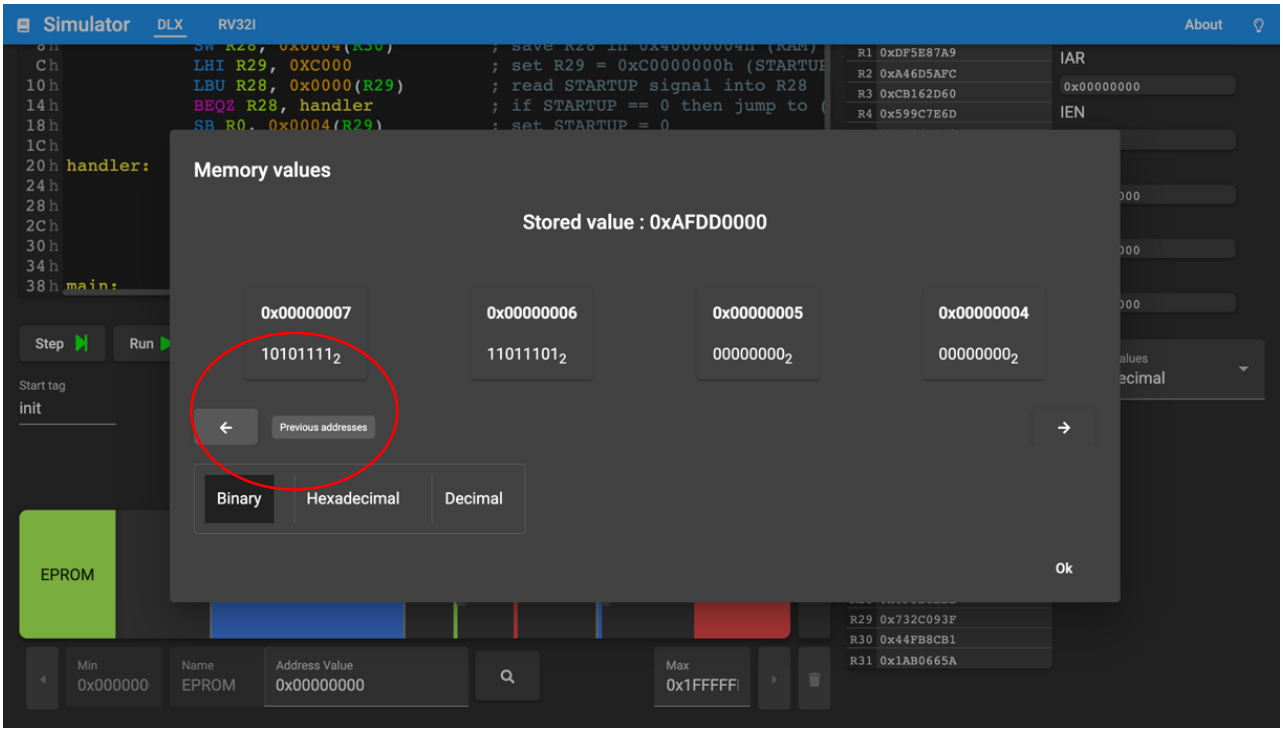

Figura 3.6

Nelle figure 3.5 e 3.6 sono evidenziati i due bottoni che permettono di muoversi avanti (Figura 3.5) e indietro (Figura 3.6) in memoria. Passando con il mouse sopra il bottone compare anche una breve descrizione della funzionalità che svolgono per favorire l'usabilità.
Se ad esempio ci troviamo nella situazione della Figura 3.5 e clicchiamo sul bottone con la freccia verso destra ("*Next addresses*") ci "si sposterà" avanti nella memoria e verranno visualizzate le successive 4 celle di memoria separate di un byte l'una dall'altra.
Ci si trova così nella situazione che vediamo in Figura 3.6 e da questa interfaccia ci si potrà spostare ancora in avanti di 4 celle di memoria oppure tornare indietro cliccando sul bottone con la freccia verso sinistra ("*Previous addresses*"). In questo modo si tornerà a visualizzare l'interfaccia delle Figura 3.5.
Questo meccanismo intuitivo e rapido permette di muoversi agilmente in memoria, senza dover ogni volta chiudere l'interfaccia di visualizzazione dei valori in memoria e dover scrivere nuovamente l'indirizzo di cui si vogliono visualizzare i valori.

3.4 Salvataggio in memoria del codice

Il codice assembly risiede in memoria e, andando quindi a leggere all'indirizzo nel quale è salvata la codifica dell'istruzione, dovremo visualizzare quest'ultima.
In particolare, nel simulatore il codice viene sempre salvato a partire dall'indirizzo 0x0000000h e l'indirizzo di ciascuna istruzione viene indicato a lato dell'editor di testo per ciascuna riga di codice.

Figura 3.7

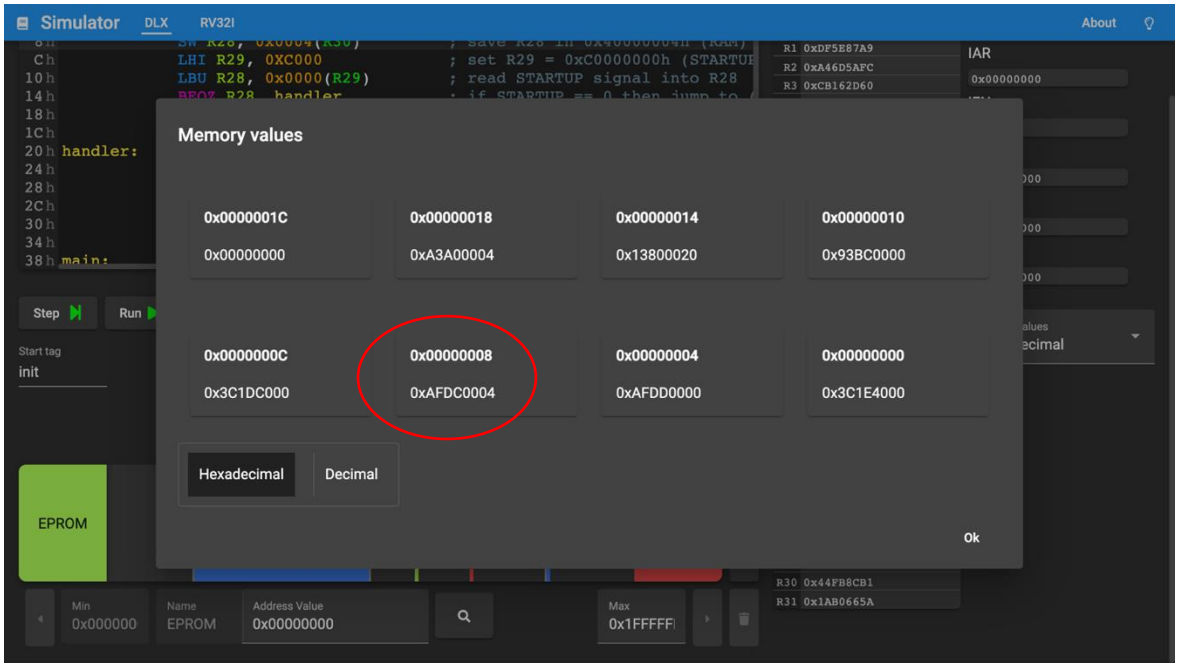

Figura 3.8

Si può osservare ad esempio in Figura 3.7 che la terza riga di codice contiene l'istruzione
"SW R28, 0x0004(R30)" e a lato è indicato che la codifica di quest'ultima è salvata all'indirizzo 0x0000008h. Si può vedere in Figura 3.8 che l'indirizzo di memoria 0x00000008h contiene la codifica dell'istruzione sopra citata.

3.4.1 Esempio codifica istruzione

Si utilizza la visualizzazione byte per byte della memoria per analizzare come viene salvata la codifica dell'istruzione *"SW R28, 0x0004(R30)"* in memoria.

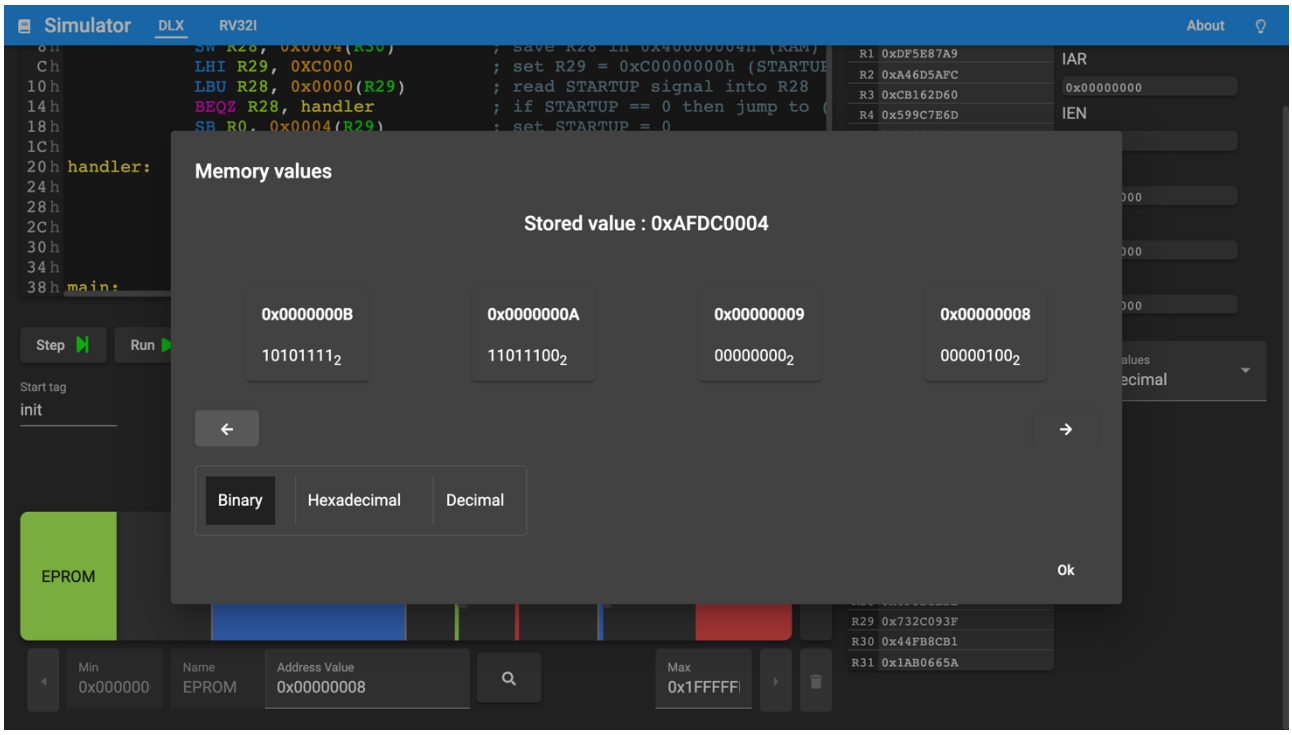

Figura 3.9

L'istruzione *Store* rientra nelle istruzioni di tipo I che, come già spiegato nel capitolo *1.2.2.1,* hanno un formato che comprende nell'ordine: un codice operativo (6 bit), un codice che rappresenta il registro destinazione (5 bit), un codice che rappresenta il registro contenente il valore da salvare in memoria (5 bit) e un immediato a 16 bit.
Il codice operativo per la SW è 101011 [5].
Il registro destinazione è il numero 30 che, codificato in binario, produce il codice 11110.
Il registro che contiene il valore da salvare è il numero 28 che, codificato in binario, produce il codice 11100.
L'immediato è 0x0004 che, codificato in binario, con 16 bit produce il codice $(0)^{13}100$.
Vediamo in Figura 3.9 che le celle di memoria contengono concatenati fra loro i codici sopra elencati, con l'ordine che va dal byte meno significativo a quello più significativo seguendo sempre la convenzione *Little Endian*.

3.5 Aggiunta Contatore

Tra le novità introdotte nel simulatore c'è la possibilità di aggiungere nello spazio di indirizzamento un contatore che può essere comandato tramite letture o scritture a specifici Chip-Select.

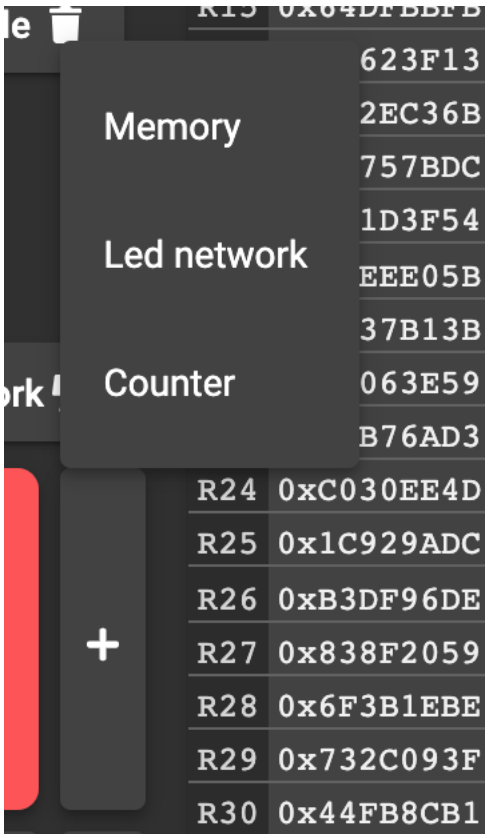

Figura 3.10

Cliccando sul tasto "+", collocato nella zona dei device, è possibile aggiungere un contatore cliccando su *"Counter"* (Figura 3.10). Comparirà quindi tra i device l'icona del contatore (Figura 3.11).

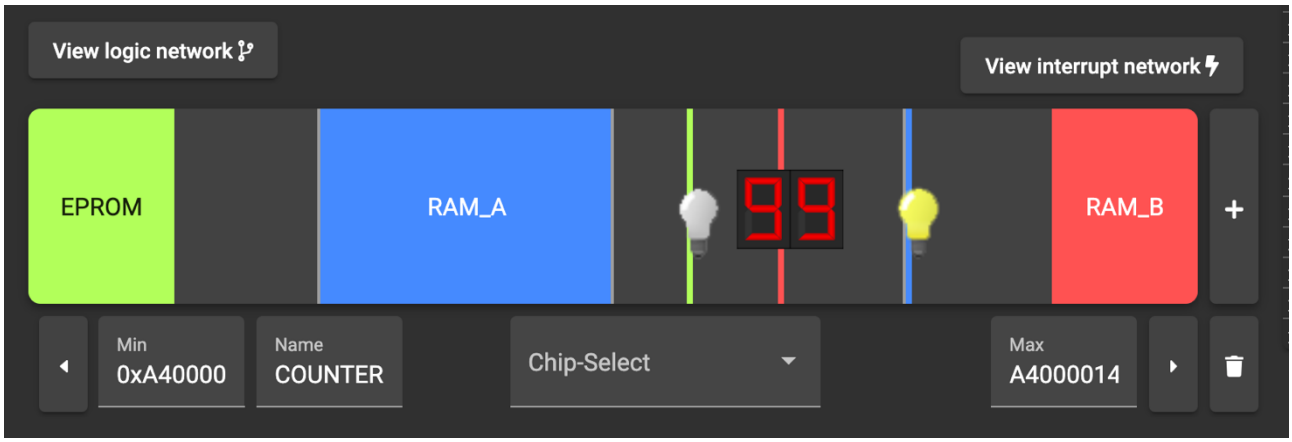

Figura 3.11

Come per le memorie e per il led è possibile scegliere gli indirizzi in cui mappare il contatore e gli indirizzi in cui mappare i chip select di quest'ultimo (Figura 3.11). Inoltre, selezionando il contatore dalla sua icona, è possibile visualizzare lo schema ai morsetti della rete logica cliccando su *"View logic network"* (Figura 3.11).

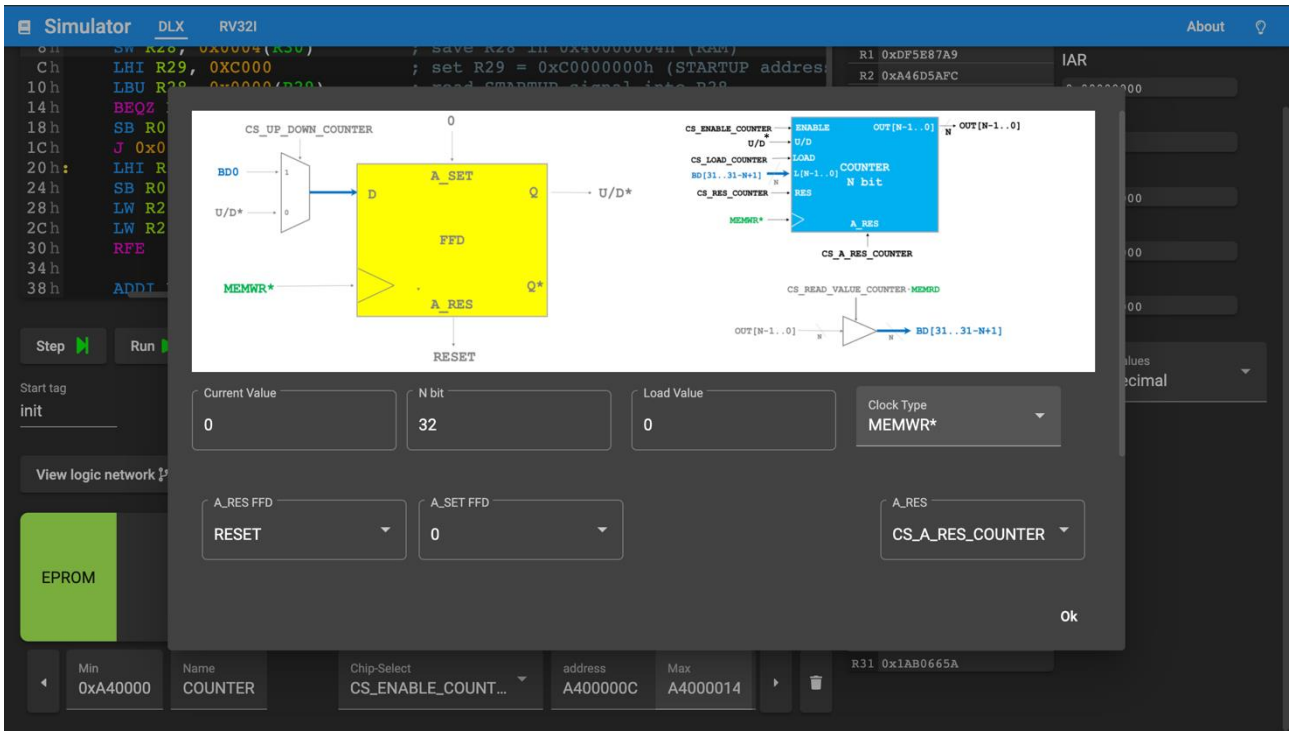

Figura 3.12

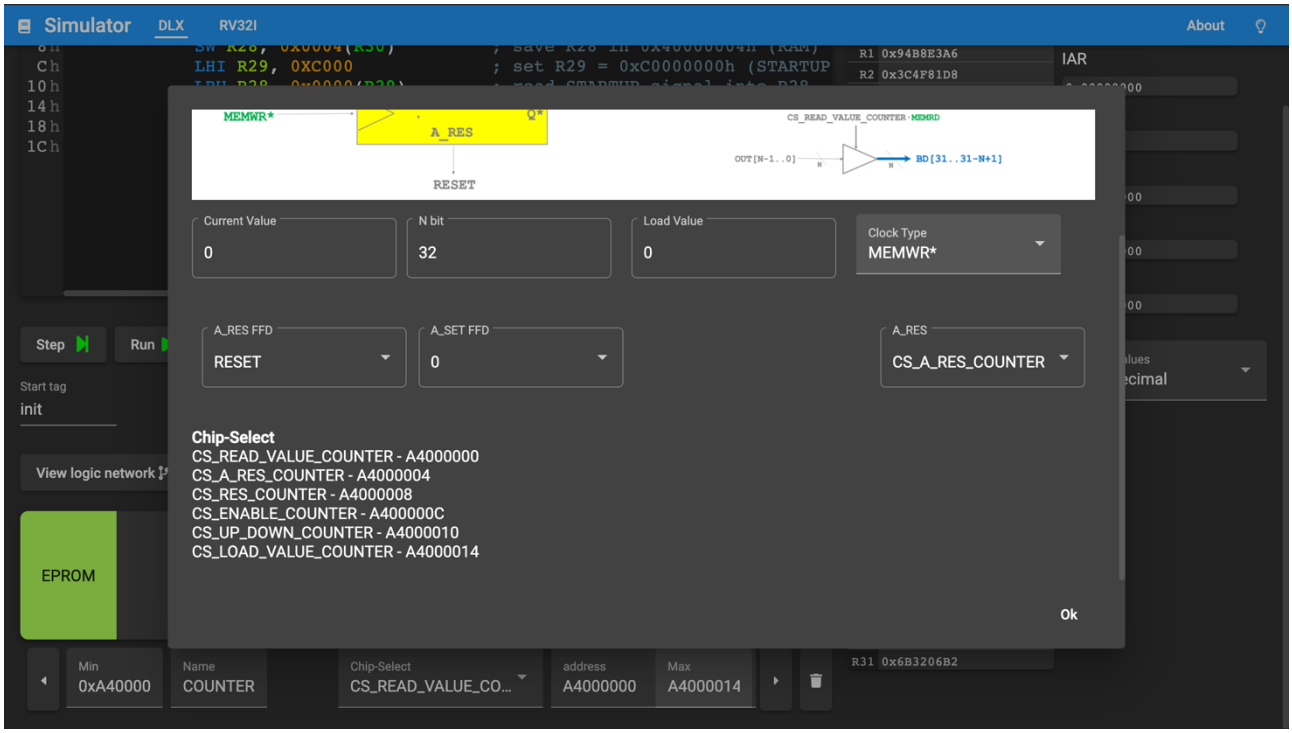

Figura 3.13

In Figura 3.12 vediamo lo schema ai morsetti del contatore e nella parte bassa dell'interfaccia una serie di campi che permettono di personalizzare i chip select, di scegliere i valori da mettere sugli ingressi e di selezionare il segnale da utilizzare come clock.
Scorrendo in basso la finestra di dialogo compare una lista dei chip select e dei relativi indirizzi in cui sono mappati (Figura 3.13).
Il campo *"N bit"* permette di selezionare la base di conteggio del contatore, mentre nel campo *"Current Value"* si può osservare il valore corrente mantenuto dal contatore.
Dal campo *"Clock type"* è possibile decidere se utilizzare il segnale MEMWR* di fine scrittura in memoria o se utilizzare MEMRD*, ossia il segnale di fine lettura. Tutti i chip select saranno quindi sincroni al segnale selezionato. Se è selezionato come clock MEMRD* le letture impartiranno il comando associato al chip select, viceversa se è selezionato MEMWR* saranno le scritture ad impartire il comando.
Di seguito analizzeremo il significato di ciascuno dei chip select:

- *CS_UP_DOWN_COUNTER*: stabilisce se il conteggio vada effettuato in avanti o all'indietro. Di default il conteggio parte in avanti ed effettuando letture o scritture all'indirizzo in cui è mappato il chip select si inverte la direzione. È necessario un flip-flop per mantenere la direzione corrente di conteggio.
- *CS_ENABLE_COUNTER*: abilita o disabilita il conteggio. Effettuando letture o scritture a questo chip select si incrementa o decrementa il valore del contatore, a seconda che il conteggio sia impostato in avanti o all'indietro.
- *CS_LOAD_COUNTER*: imposta il valore di conteggio ad un valore fornito dall'esterno sugli ingressi L [N-1..0]. Il valore degli ingressi L è selezionabile dal campo *"Load Value"* dov'è possibile scrivere il valore che si desidera immettere.
- *CS_RES_COUNTER*: resetta il contatore in modo sincrono.
- *CS_A_RES_COUNTER*: resetta il contatore in modo asincrono.
- *CS_READ_VALUE_COUNTER*: effettuando una lettura a questo chip select si ottiene il valore corrente memorizzato nel contatore. È ovviamente messo in and logico con il segnale di MEMRD perché il valore può essere ottenuto solamente tramite una lettura; una scrittura a questo chip select non avrebbe alcun senso.

3.5.1 Esempi di utilizzo del contatore

Si osserva ora qualche semplice esempio di codice assembly che utilizza il contatore e si analizzano i comportamenti ottenuti.

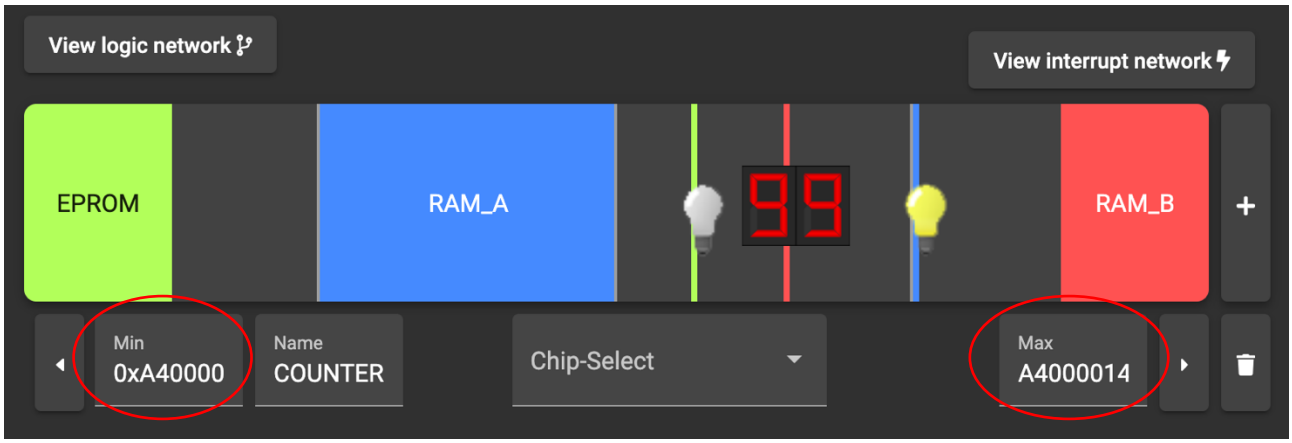

Figura 3.14

Il contatore è mappato (Figura 3.14) nel range di indirizzi che va da 0xA4000000 a 0xA4000014, mentre gli indirizzi dei chip select sono quelli in Figura 3.13.

3.5.1.1 Esempio 1

```
 0h          LHI R30, 0xA400          ;
 4h          SW R29, 0x000C(R30)      ; increments counter
 8h          SW R29, 0x000C(R30)      ; increments counter
 Ch          SW R29, 0X0010(R30)      ; reverse counting
10h          SW R29, 0x000C(R30)      ; decrements counter
14h          SW R29, 0x0008(R30)      ; syncronous reset
```

Figura 3.15 – Codice esempio 1

La prima riga di codice (*0h*) prepara il registro R30 per essere utilizzato successivamente nelle store.
La seconda e terza istruzione (*4h, 8h*) effettuano delle scritture al chip select *CS_ENABLE_COUNTER*. Dato che è selezionato come clock type il MEMWR* (Figura 3.16) verrà impartito il comando di enable ed essendo di default impostato il conteggio in avanti, il contatore verrà incrementato di due.

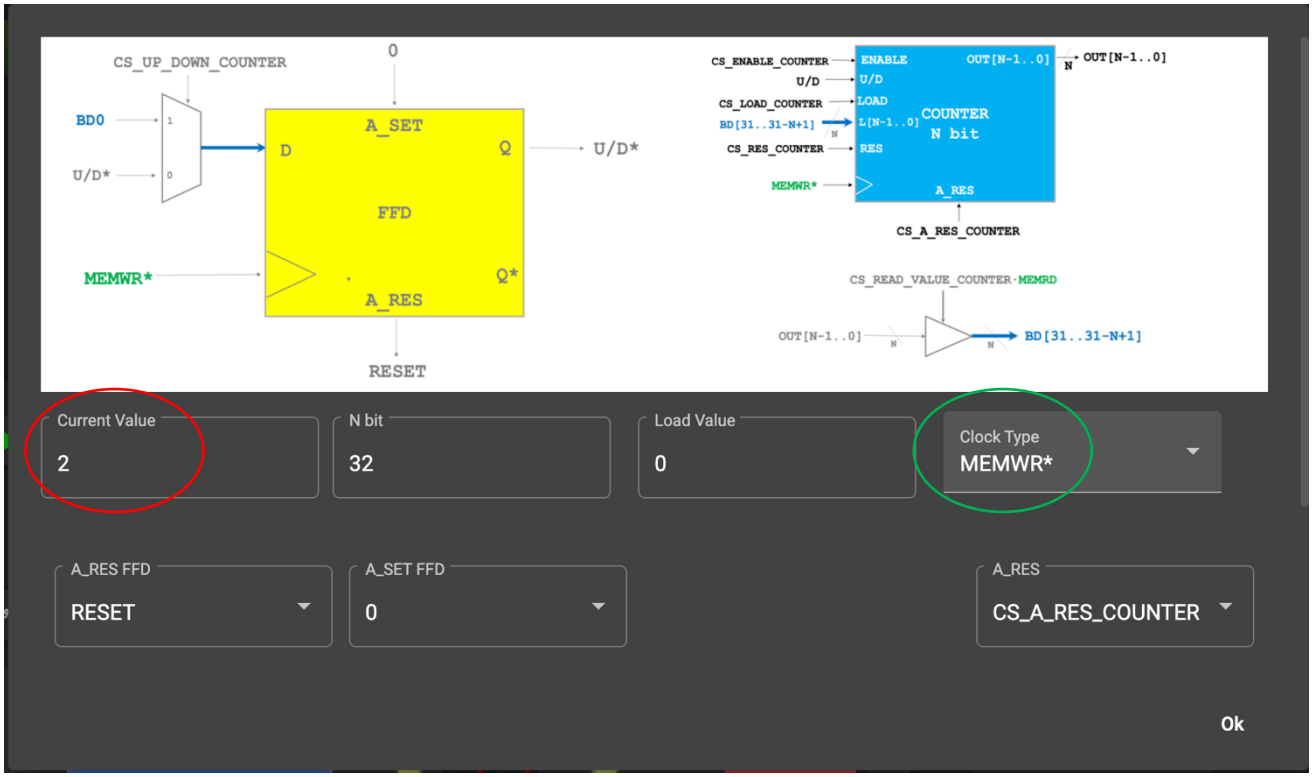

Figura 3.16 - Contatore incrementato di due e clock type impostato a MEMWR*

La quarta istruzione (*Ch*) effettua una scrittura a *CS_UP_DOWN_COUNTER* invertendo la direzione del conteggio, la successiva scrittura (*10h*) al chip select *CS_ENABLE_COUNTER* decrementerà quindi il valore corrente del contatore di uno.
L'ultima istruzione effettua una scrittura al chip select *CS_RES_COUNTER* che resetta in modo sincrono il contatore portando il valore corrente a 0.

3.5.1.2 Esempio 2

```
0h    LHI R30, 0xA400 ;
4h    LW R29, 0x0014(R30) ;
8h    SW R29, 0x000C(R30) ;
Ch    LW R29, 0x000C(R30) ;
10h   LW R28, 0X0000(R30) ;
```

Figura 3.17 – Codice esempio 2

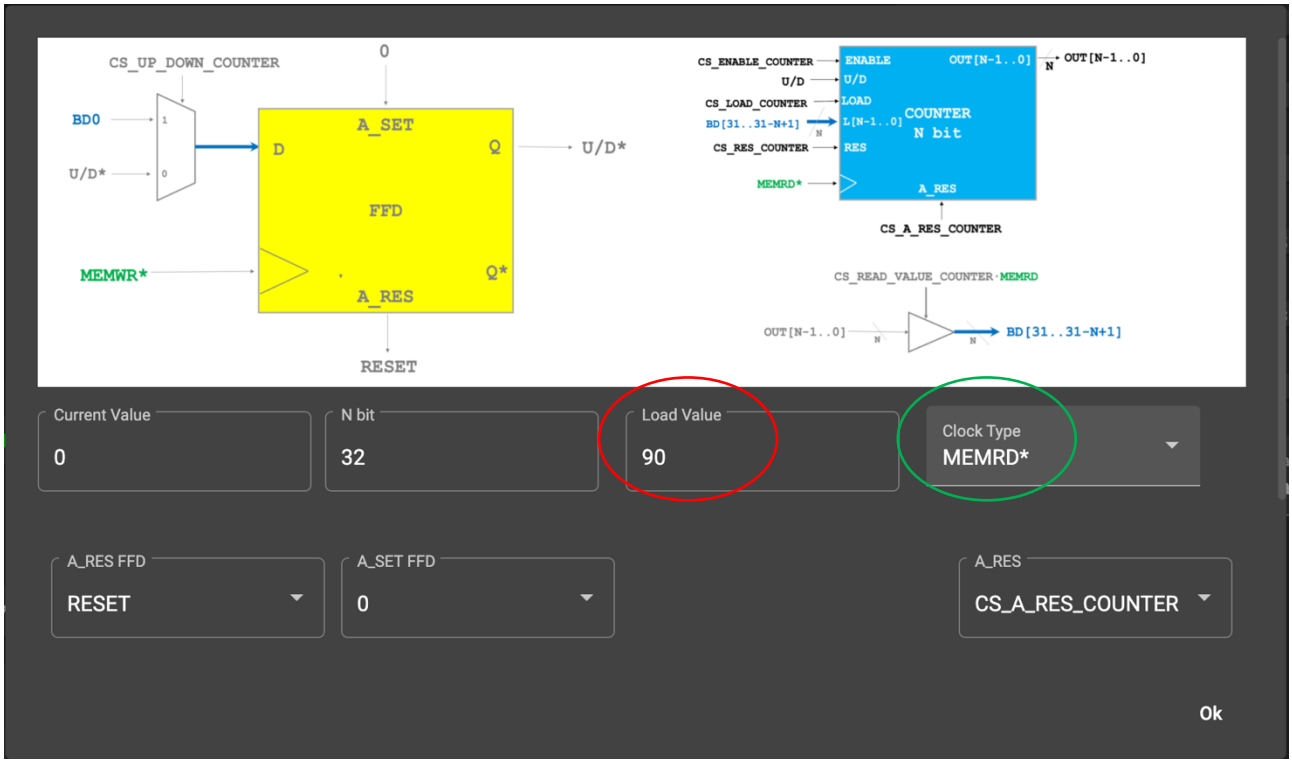

Figura 3.18

In questo esempio è stato selezionato come clock type MEMRD*, quindi per attivare un comando bisognerà effettuare una lettura al chip select associato. Notiamo anche in Figura 3.18 che abbiamo impostato sugli ingressi L il valore 90.
La prima riga di codice (*0h*) prepara il registro R30 per essere utilizzato successivamente nelle load.
La seconda riga di codice (*4h*) carica nel contatore il valore presente sugli ingressi L effettuando una lettura al chip select *CS_LOAD_VALUE_COUNTER*.

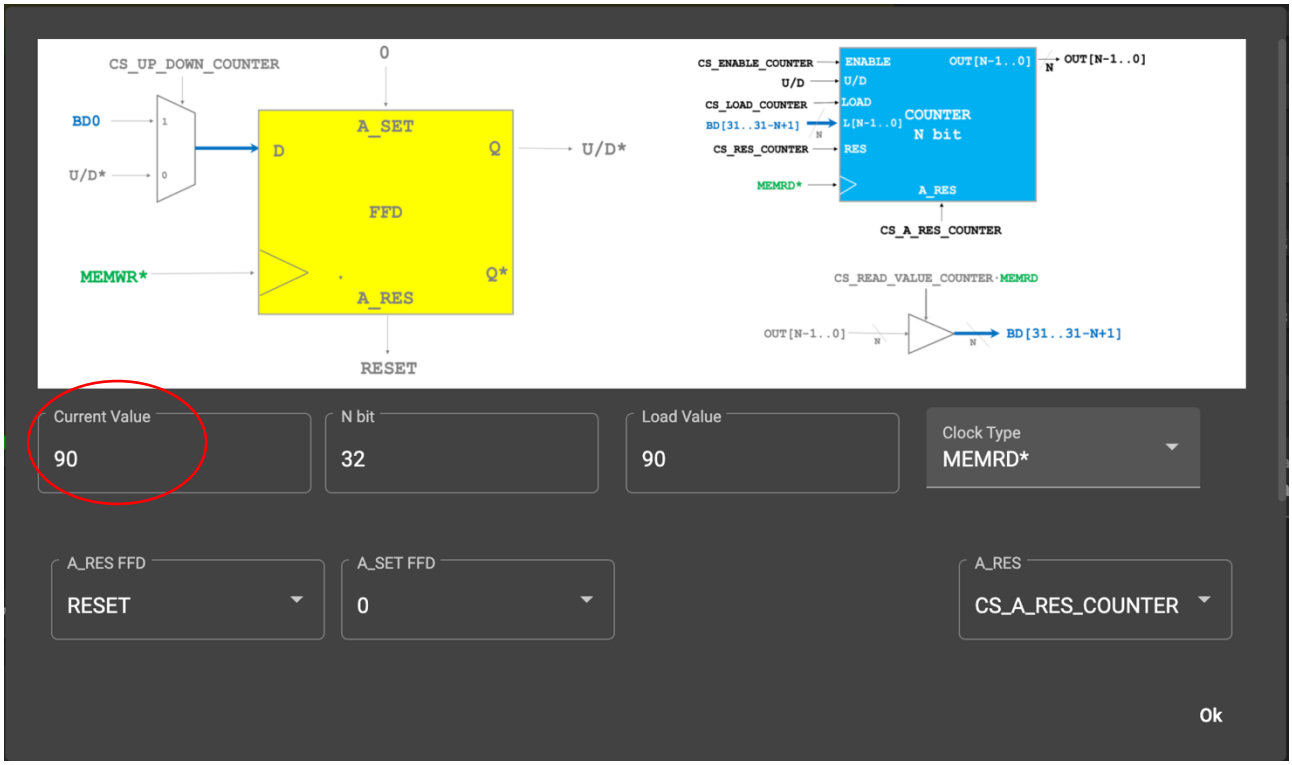

Figura 3.19

In Figura 3.19 notiamo come il valore corrente del contatore sia stato inizializzato con quello presente sugli ingressi L.
La terza istruzione (*8h*) non produce alcun effetto perché, essendo impostato come clock MEMRD*, il comando viene eseguito in corrispondenza delle letture e non delle scritture.
La quarta istruzione (*Ch*) invece effettua una lettura al *CS_ENABLE_COUNTER* incrementando il valore del contatore di uno.
L'ultima istruzione (*10h*) effettua una lettura al *CS_READ_VALUE_COUNTER* andando a salvare il valore del contatore nel registro R28.

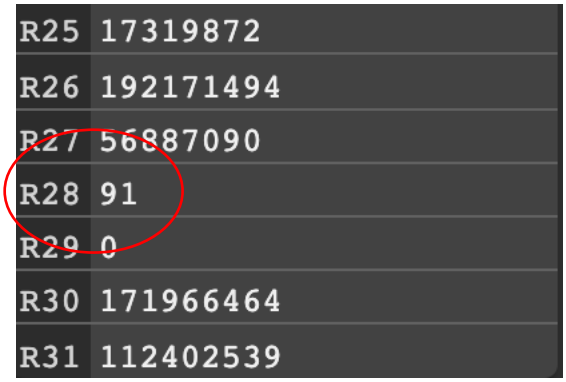

Figura 3.20

Si può osservare in Figura 3.20 che il registro R28 contiene il valore 91 che è il valore iniziale del contatore incrementato di 1.

4 DEPLOYMENT

Al fine di rendere fruibile il simulatore a ciascuno studente da remoto, quest'ultimo è stato ospitato su un server del Dipartimento di Informatica – Scienza e Ingegneria dell'Università di Bologna.
Si noti che per eseguire la maggior parte delle operazioni descritte in seguito sono necessari i privilegi di amministratore sul server.

4.1 Configurazione web server Apache

Innanzitutto è stato installato sulla macchina il web server Apache. In primis bisogna aggiornare l'indice del pacchetto e poi procedere all'installazione.

sudo apt update
sudo apt install apache2

Una volta finita l'installazione il servizio apache2 è attivo e si può controllare lo stato con il comando *sudo systemctl status apache2*.

```
[root@dlx-simulator:~# systemctl status apache2
● apache2.service - The Apache HTTP Server
     Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor prese>
     Active: active (running) since Wed 2021-06-23 06:58:45 UTC; 2 months 5 day>
       Docs: https://httpd.apache.org/docs/2.4/
    Process: 2247974 ExecReload=/usr/sbin/apachectl graceful (code=exited, stat>
   Main PID: 1375833 (apache2)
      Tasks: 55 (limit: 2280)
     Memory: 12.0M
     CGroup: /system.slice/apache2.service
             ├─1375833 /usr/sbin/apache2 -k start
             ├─2247978 /usr/sbin/apache2 -k start
             └─2247979 /usr/sbin/apache2 -k start
```

Figura 4.1 - Servizio apache2 attivo

In seguito, è stato configurato il firewall per permettere al servizio apache2 di svolgere i suoi compiti e in particolare di utilizzare la porta 80:

sudo ufw allow 'Apache Full'

```
root@dlx-simulator:~# sudo ufw status
Status: active

To                         Action      From
--                         ------      ----
80                         ALLOW       Anywhere
22                         ALLOW       Anywhere
4200                       ALLOW       Anywhere
Apache Full                ALLOW       Anywhere
80 (v6)                    ALLOW       Anywhere (v6)
22 (v6)                    ALLOW       Anywhere (v6)
4200 (v6)                  ALLOW       Anywhere (v6)
Apache Full (v6)           ALLOW       Anywhere (v6)
```

Figura 4.2 - Stato del firewall

È stato quindi modificato il file */etc/apache2/apache2.conf* impostando i giusti permessi di accesso alla directory */var/www.* La Figura 4.3 mostra le righe del file apache2.conf che sono state modificate.

```
<Directory /var/www/>
        Options Indexes FollowSymLinks
        AllowOverride All
        Require all granted
</Directory>
```

Figura 4.3

Una volta apportate le modifiche al file, è stato riavviato il servizio apache2 con il seguente comando:

sudo systemctl restart apache2

4.2 Configurazioni per la sicurezza

Affinchè si possano prevenire eventuali attacchi sono state fatte alcune configurazioni mirate ad aumentare la sicurezza del web server Apache.

4.2.1 Configurare la pagina d'errore

È stata configurata la pagina d'errore affinché non venga mostrata quella predefinita di Apache, ma che si venga reindirizzati alla pagina principale dell'applicazione. L'utilizzo delle pagine di Apache è sconsigliato per due motivi: In primo luogo, esse rappresentano un potenziale problema di sicurezza in quanto possono offrire ad un attaccante delle informazioni sul web server; inoltre, un messaggio di errore standard viene in genere percepito dall'utente come sintomo di grave malfunzionamento del sistema [10].
A questo scopo è stato modificato il file /etc/apache2/sites-enabled/000-default.conf come vediamo in Figura 4.4. In particolare, sulla riga "*ErrorDocument 404 …"* deve comparire l'indirizzo della pagina che vogliamo venga visualizzata in caso di errore. Nel nostro caso abbiamo scelto la pagina principale del simulatore che ha come indirizzo *"/simulator/index.html"*.

```
<VirtualHost *:80>
        # The ServerName directive sets the request scheme, hostname and port that
        # the server uses to identify itself. This is used when creating
        # redirection URLs. In the context of virtual hosts, the ServerName
        # specifies what hostname must appear in the request's Host: header to
        # match this virtual host. For the default virtual host (this file) this
        # value is not decisive as it is used as a last resort host regardless.
        # However, you must set it for any further virtual host explicitly.
        #ServerName www.example.com

        ServerAdmin webmaster@localhost
        DocumentRoot /var/www/html
        ErrorDocument 404 /simulator/index.html

        # Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
        # error, crit, alert, emerg.
        # It is also possible to configure the loglevel for particular
        # modules, e.g.
        #LogLevel info ssl:warn

        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined

        # For most configuration files from conf-available/, which are
        # enabled or disabled at a global level, it is possible to
        # include a line for only one particular virtual host. For example the
        # following line enables the CGI configuration for this host only
        # after it has been globally disabled with "a2disconf".
        #Include conf-available/serve-cgi-bin.conf
</VirtualHost>

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet
```

Figura 4.4 – File /etc/apache2/sites-enabled/000-default.conf

Una volta modificato il file è stato riavviato il servizio apache2 per far sì che le modifiche apportate diventassero effettive.

4.2.2 Disabilitazione del directory listing

Di default, se si richiede un URL per la quale non è presente un file index, Apache restituisce una pagina con la lista dei file all'interno della directory. Ciò fornisce ad eventuali attaccanti informazioni sulla struttura delle directory servite dal nostro web server. Si è ricorso quindi alla direttiva *Indexes,* riportandola all'interno della directory da proteggere che nel nostro caso era la directory /var/www/html.

```
<Directory /var/www/html>
        Options -Indexes
</Directory>
```

Figura 4.5

In Figura 4.5 vediamo la direttiva scritta all'interno del file di configurazione /etc/apache2/apache2.conf. Una volta modificato il file è stato riavviato il servizio apache2 per far sì che le modifiche apportate diventassero effettive.

4.2.3 Nome e versioni di Apache nascosti

Fornire nome e versione del server ad un attaccante può permettergli di sfruttare vulnerabilità già note per quella macchina e quella versione del software. Digitando particolari URL si possono ottenere pagine di default di Apache che forniscono queste informazioni come vediamo in Figura 4.6.

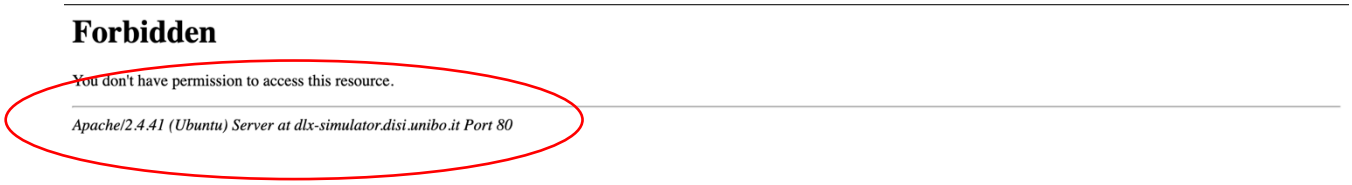

Figura 4.6

Per disabilitare questo comportamento si può configurare Apache con le direttive *ServerSignature Off* e *ServerTokens Prod* che vanno scritte nel file /etc/apache2/conf-available/security.conf. Dopo questa configurazione le pagine restituite da Apache saranno come quelle in Figura 4.7, senza nessuna indicazione sulla versione di Apache e sul nome del dominio del server.

Figura 4.7

4.3 Deployment del simulatore sul server

Dopo queste configurazioni il server è pronto per ospitare il progetto del simulatore. Tutti i passi descritti di seguito sono stati eseguiti ogni volta che è stato fatto il deployment di una nuova versione del simulatore e dovranno essere ripetuti in futuro da chi vorrà apportare delle modifiche a quest'ultimo.
Va inoltre specificato che sono state mantenute due repository GitHub: una contenente l'intero progetto [12], dalla quale può essere effettuato un "branch" da futuri studenti che lavoreranno su questo progetto; una contenente solo i file compilati pronti per essere caricati sul server, che viene utilizzata per il deployment effettuando un "clone" della repository direttamente dal server.

In primis è necessario compilare l'intero progetto in una serie di file statici che potranno essere caricati sul server tramite il comando

ng build - - prod - - base-href=/simulator/

L'opzione - - prod crea una versione ottimizzata dell'applicazione, mentre l'opzione - - base-href specifica l'url base dell'applicazione che viene compilata, nel nostro caso è il nome della cartella che contiene i file compilati all'interno del server. I file compilati vengono salvati nella cartella *"dist"* presente all'interno del progetto Angular. Dopo aver aggiornato la repository GitHub contenente i file compilati vanno quindi scaricati i file sul server. Bisognerà prima spostarsi nella cartella /var/www/html, la directory che conterrà i file compilati del simulatore, e scaricare direttamente in quella posizione il progetto da GitHub:

cd /var/www/html

git clone https://github.com/FilippoComastri/DLX-simulator-dist.git

Sul server comparirà, nella stessa posizione dalla quale è stato lanciato il comando, una cartella *DLX-simulator-dist* che contiene tutti i file compilati. A questo punto basta modificare il nome della cartella in "simulator" con il comando

mv DLX-simulator-dist simulator

e l'applicazione sarà disponibile digitando sul browser l'url del simulatore [7].

5 SVILUPPI FUTURI

La parte del simulatore nella quale possono essere apportate significative migliorie è quella che riguarda la simulazione del clock e dei cicli di bus in lettura e scrittura. Ad oggi è solamente possibile selezionare in modo statico quale tipo di segnale collegare al clock, ma i segnali di MEMRD e MEMWR essendo simulati non sono prodotti da veri cicli di bus. Sarebbe utile approfondire nel simulatore un meccanismo in grado di simulare i cicli di bus in lettura e scrittura ogni qualvolta avvengano delle load o store in memoria, così da rendere molto più realistici e fedeli alla realtà i segnali che vengono utilizzati come clock.
Altra novità interessante da introdurre è la possibilità di aprire una finestra nella quale visualizzare in tempo reale le forme d'onda generate da ciascun accesso alla memoria, ovvero quando avvengono delle letture o scritture.
Con questa novità sarebbe possibile anche inserire nel progetto la gestione di periferiche I/O e simularne i processi di lettura e scrittura.
Inoltre sarebbe utile affiancare all'applicazione un tool o un editor che permetta di creare delle reti logiche più complesse e articolate oltre a quelle già presenti nel simulatore.

6 CONCLUSIONI

Lo scopo del progetto di estendere e migliorare il simulatore è stato raggiunto e tutte le novità introdotte funzionano correttamente e svolgono il compito per il quale sono state progettate.
Sono stati inoltri corretti alcuni piccoli errori sorti durante l'utilizzo del simulatore e limato qualche dettaglio:

- Nel campo *"Address value"* è stato inserito un controllo sugli indirizzi inseriti: è possibile scrivere gli indirizzi solo nel formato *"0x"* e se si inserisce un indirizzo nel quale non è mappato alcun dispositivo compare un messaggio d'errore.
- È stato reso uniforme il formato delle store con quello presentato nelle slide del corso Calcolatori Elettronici-T tenuto da Stefano Mattoccia.
- È stato corretto un errore che si presentava quando venivano eseguite le istruzioni J, BEQZ, BNEZ: veniva sempre effettuato un salto assoluto mentre l'istruzione prevede un salto relativo all'indirizzo di memoria in cui si trova l'istruzione eseguita.
- Ora le memorie e i registri vengono inizializzati con valori casuali e non sempre con il valore "0" come avveniva precedentemente.

Tutte le modifiche sono state apportate seguendo le linee guida progettuali scelte dagli studenti che hanno lavorato al simulatore prima di me, cercando inoltre di favorire future modifiche e preservare scalabilità e modularità.
Gli argomenti trattati durante questa tesi di laurea sono stati principalmente quelli già visti durante il corso di Calcolatori Elettronici T, il che mi ha permesso di approfondire e di consolidare le mie conoscenze.
Ho inoltre approfondito le mie conoscenze nel campo delle tecnologie web avendo avuto l'occasione di lavorare con Angular, un framework mai utilizzato in passato, e di utilizzare le tecnologie imparate a lezione in un progetto di discrete dimensioni.

BIBLIOGRAFIA

[1] Alessandro Foglia - “Progetto di un simulatore di RISC-V per scopi didattici”, Tesi di laurea AA 2018/19

[2] Fabrizio Maccagnani - “Progetto di un simulatore di DLX per scopi didattici”, Tesi di laurea AA 2018/19

[3] Federico Pomponii - “Sviluppo di un simulatore DLX per scopi didattici”, Tesi di laurea AA 2019/20

[4] Materiale del corso di Calcolatori Elettronici T, Ingegneria Informatica, Università di Bologna, tenuto da Stefano Mattoccia. Sezione 03 Linguaggio Macchina http://vision.deis.unibo.it/~smatt/DIDATTICA/Calcolatori_Elettronici_T/PDF/03_Linguaggio_macchina.pdf

[5] University of Crete – Computer Science Department, Codici Operativi per le istruzioni Dlx
https://www.csd.uoc.gr/~hy425/2002s/dlxmap.html

[6] Html.it – “Introduzione a Typescript”
https://www.html.it/pag/55620/introduzione-a-typescript/

[7] Dipartimento di Informatica – Scienza e Ingegneria, Università di Bologna, Simulatore DLX
http://dlx-simulator.disi.unibo.it/simulator/dlx

[8] mrw.it – Web server: cos’è e come funziona
https://www.mrw.it/linux/web-server-come-funziona_12156.html

[9] blog.hostingperte.it - Cos’è Apache? Analisi del Web Server Apache
https://blog.hostingperte.it/cose-apache-analisi-del-web-server-apache/

[10] html.it – Guida Apache, Pagine di Errore
https://www.html.it/pag/31982/messaggi-di-errore/

[11] mrw.it – Introduzione ad Angular CLI
https://www.mrw.it/javascript/introduzione-angular-cli_12717.html

[12] Filippo Comastri – Repository GitHub, Progetto completo
https://github.com/FilippoComastri/DLX-RISCV-simulator.git

[13] Filippo Comastri – Repository GitHub, Progetto con file compilati per il deployment
https://github.com/FilippoComastri/DLX-simulator-dist.git