



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**

**SIMULATORE DEL PROCESSORE DLX PER SCOPI DIDATTICI**

**CANDIDATO**

Gabriele Piazzì

**RELATORE**

Stefano Mattoccia

Anno Accademico 2021/2022



# Indice

<b>Introduzione.....</b>	<b>4</b>
<b>1.1 Architettura DLX.....</b>	<b>5</b>
<b>1.1.1 Modalità di accesso alla memoria .....</b>	<b>6</b>
<b>1.1.2 Istruzioni DLX.....</b>	<b>7</b>
<b>1.1.3 Protocollo Handshake .....</b>	<b>8</b>
<b>Strumenti Utilizzati .....</b>	<b>10</b>
<b>2.1 Angular.....</b>	<b>10</b>
<b>2.2 Sistema di controllo delle versioni - Git .....</b>	<b>12</b>
<b>Correzione bug.....</b>	<b>13</b>
<b>Nuove funzionalità.....</b>	<b>15</b>
<b>4.1 Aggiunta Input-port .....</b>	<b>16</b>
<b>4.1.1 Esempi di funzionamento Input-port .....</b>	<b>19</b>
<b>4.2 Evidenziazione registri .....</b>	<b>21</b>
<b>Sviluppi futuri.....</b>	<b>23</b>
<b>Conclusione.....</b>	<b>24</b>
<b>Bibliografia .....</b>	<b>26</b>

# Capitolo 1

## Introduzione

Lo scopo di questa tesi di laurea è quello di estendere e 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] , Filippo Comastri [4] e Umberto Laghi [5].

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.

La prima parte della tesi riguarda la correzione di bug riscontrati nell'utilizzo e che, alle volte, non rispecchiano perfettamente il vero funzionamento. La seconda parte prevede l'inserimento di una simulazione di comunicazione e scambio di dati tra il processore DLX e un'unità esterna e di rendere più leggibile lo stato sei registri.

## 1.1 Architettura DLX

DLX è un'architettura per microprocessori sviluppata da John L. Hennessy e David A. Patterson, partendo dall'architettura MIPS e semplificandola per scopi didattici. Lo studio del DLX è molto diffuso, perché il suo ISA è molto simile a quello che viene utilizzato nei processori basati su RISC-V. L'ISA del DLX prevede 32 registri da 32bit ciascuno (R0-R31): R0 è cablato a 0, R31 viene usato per salvare l'indirizzo di ritorno per le istruzioni di salto come JAL (Jump And Link) e JALR (Jump And Link Register), mentre tutti gli altri registri sono general purpose (GPR).

Inoltre, sono presenti anche altri registri speciali, sempre a 32 bit, a cui il programmatore non può accedere. Questi registri sono: il Program Counter (PC), che memorizza l'indirizzo dell'istruzione successiva a cui il processore effettuerà il fetch; l'Interrupt Address Register (IAR), che contiene l'indirizzo di ritorno al main nel caso della ricezione di una interruzione o interrupt;

il Memory Address Register (MAR), che mantiene l'indirizzo di memoria dove vogliamo andare a leggere o scrivere; il Memory Data Register (MDR), che contiene il dato appena letto o da scrivere.

Come si può notare dall'immagine sotto riportata, il DLX comunica con il resto del sistema tramite bus, poiché basato sul modello di Von Neumann.

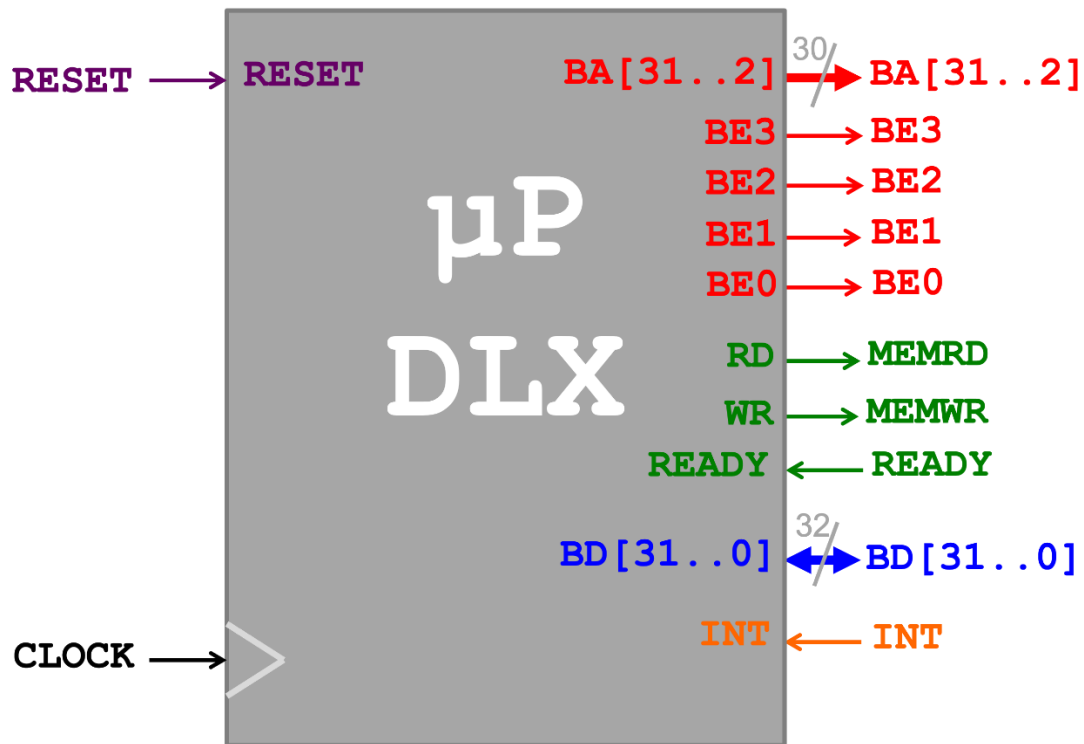


Figure 1: DLX ai "morsetti"

### 1.1.1 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.

$$\text{Indirizzo} = \text{costante} + \text{registro}$$

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.

## 1.1.2 Istruzioni DLX

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

L'unica costante tra questi tre tipi di istruzioni è che l'OpCode sia posto nei 6 bit più significativi, questo è stato fatto per semplificare la progettazione dell'Unità di Controllo, che così conosce già a priori la posizione dell'OpCode.

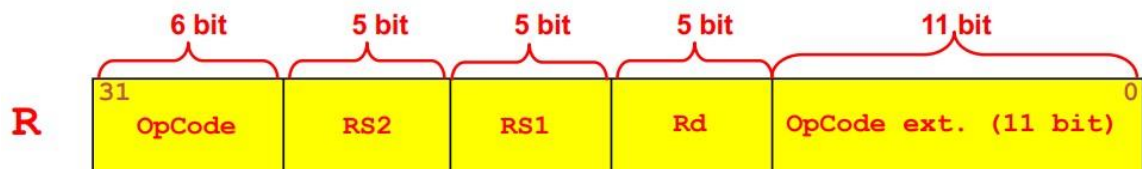
### - Tipo I



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).

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.

### - Tipo R



Il gruppo R riguarda tutte le istruzioni che operano tra registri.

RS1 ed RS2 sono i registri sorgente da cui vengono prelevate le stringhe di bit, OpCpde extension è una stringa di 11 bit utilizzabile per fare un'ulteriore differenziazione tra le istruzioni nel caso in cui l'ISA venga esteso.

- Tipo J



Le istruzioni di tipo J includono Jump (J), Jump And Link (JAL), TRAP, e Return From Exception (RFE). 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.

### 1.1.3 Protocollo Handshake

Il protocollo handshake è un metodo utilizzato per sincronizzare le comunicazioni e lo scambio di dati tra la CPU e un dispositivo esterno.

Esistono due tipi di protocolli, uno in input che permette alla CPU di ricevere un dato dall'esterno, e uno in output che consente alla CPU di inviare dati all'esterno.

Questi protocolli vengono realizzati tramite un'interfaccia chiamata input/output port che si pone tra il processore e l'unità esterna.

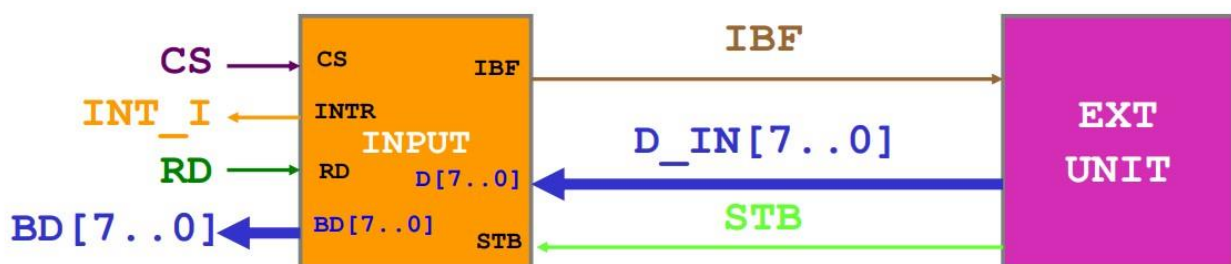


Figure 2: Input Port



Nel momento in cui l'unità esterna vuole inviare un dato alla CPU, asserisce il segnale STB ad 1, questo viene ricevuto dall'input port che contemporaneamente asserirà il segnale IBF (Input Buffer Full) ad 1, comunicando all' unità esterna che il buffer è pieno e fintantoché non verrà svuotato, essa dovrà bloccare l'invio di altri dati all'interfaccia di input.

Una volta terminato di scrivere il dato l'UE riporterà il segnale STB a 0 e questo comporterà il fronte di salita di INT\_I da parte dell'input-port che provvederà a comunicare alla CPU che il dato è pronto per essere gestito. Arrivati a tal punto, il processore, pronto per la gestione dell'interrupt, inizierà il ciclo di lettura, portando MEMRD a 1 e INT\_I a 0. Terminato il ciclo di lettura e salvato il dato in memoria, il segnale MEMRD tornerà a 0 e questo comporterà il fronte di discesa di IBF, cosicché l'interfaccia in input possa ricevere un nuovo dato.

Vengono rappresentate le forme d'onda dei segnali che caratterizzano il protocollo di Handshake.

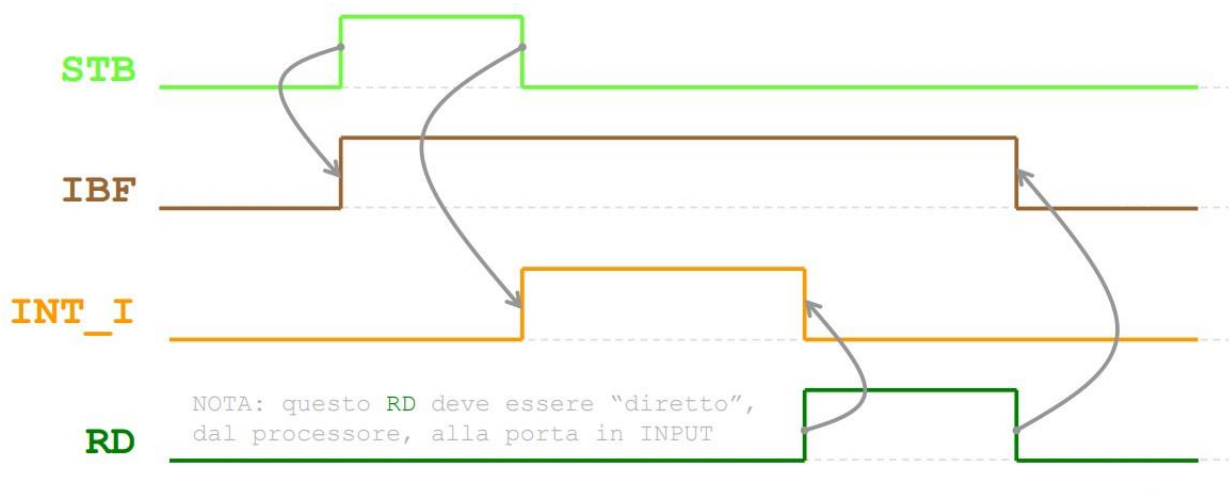


Figure 3: forme d'onda handshake

## Capitolo 2

### Strumenti Utilizzati

#### 2.1 Angular

Angular 2+, o solo Angular, è un framework open source per lo sviluppo di applicazioni web, e costituisce l'evoluzione di AngularJS, sviluppato principalmente da Google. Questa tecnologia permette di progettare e implementare progetti strutturati per la realizzazione di interfacce utente, con immediati vantaggi in termini di robustezza del codice, testabilità e manutenibilità, creando applicazioni che sono anche veloci e performanti. Angular mette a disposizione un ambiente per la creazione veloce di app, moduli, componenti e il supporto degli IDE e i numerosi plugin facilitano la stesura del codice. Il linguaggio principalmente usato per sviluppare con questo framework è TypeScript.

TypeScript è un linguaggio di programmazione open source sviluppato da Microsoft. Più nello specifico, TypeScript è un superset di JavaScript, che aggiunge tipi, classi, interfacce e moduli opzionali al JavaScript tradizionale. Si tratta sostanzialmente di una estensione di JavaScript.

TypeScript è un linguaggio tipizzato, ovvero aggiunge definizioni di tipo statico: i tipi consentono di descrivere la forma di un oggetto, documentandolo meglio e consentendo a TypeScript di verificare che il codice funzioni correttamente.

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.

Nel progetto è stato utilizzato il comando *ng build* che compila un'applicazione in una directory predefinita denominata *dist/*, che 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 *--configuration production*, che sostituisce il deprecato *--prod.* (production) del comando *ng build*.

In questo caso Angular CLI impiega un'ottimizzazione, ovvero viene utilizzato il meccanismo di compilazione AOT (Ahead-of-Time Compilation) che converte il codice HTML e TypeScript in codice javascript 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 che non è più necessario.

Un altro comando utilizzato *ng serve* che consente di visualizzare in fase di sviluppo un'anteprima della nostra applicazione nel browser. Infatti, eseguendo *ng serve*, viene creato un web server locale all'indirizzo predefinito *http://localhost:4200*, che è comunque possibile personalizzare attraverso le opzioni *--host* e *--port*.

Per le icone è stato utilizzato Clip Art che permette di scaricare immagini gratuitamente.

## 2.2 Sistema di controllo delle versioni - Git

Git è un sistema di controllo delle versioni distribuito, che permette di monitorare e gestire le modifiche al codice del software. Questi sono strumenti software che aiutano i team a gestire le modifiche apportate al codice sorgente nel tempo.

Il software di controllo delle versioni tiene traccia di ogni modifica apportata al codice in un tipo speciale di database. Se viene commesso un errore, gli sviluppatori possono tornare indietro nel tempo e mettere a confronto le versioni precedenti del codice per correggere l'errore riducendo al minimo le interruzioni per tutti i membri del team.

Git consente inoltre di sottoporre a commit le nuove modifiche, di eseguire il merge, di fare un confronto con le versioni precedenti e di creare branch.

Quest'ultima funzionalità fornisce un ambiente isolato per ogni modifica da apportare alla base di codice, cosicché il branch principale mantenga sempre codice di qualità di produzione.

Dunque, non esiste una repository centralizzata del progetto del simulatore DLX ma ogni utente mantiene una propria repository locale e la sincronizzazione avviene scambiandosi delle patch tra i vari utenti.

## Capitolo 3

### Correzione bug

Come primo passo del progetto è stato effettuato un test del simulatore, per approfondirne il funzionamento e rilevarne eventuali bug nel codice.

Sono stati testati tutti i comandi che si trovano nella sezione documentazione, per controllare che le istruzioni funzionassero correttamente, ed è stato rilevato un problema nei comandi relativi all'accesso in memoria da parte del simulatore, ovvero nelle istruzioni di load e store. Queste due operazioni permettono di leggere/scrivere un dato in memoria di lunghezza 8 bit (byte), 16 bit (half-word), oppure 32 bit (word). Il problema è dovuto dal fatto che viene effettuato un controllo errato da parte della funzione di accesso in memoria, sui valori ricevuti in ingresso. Quest'ultime ricevono un registro di destinazione (Rs1), che viene calcolato dalla somma di un registro di partenza (Rs2) a 32 bit e un immediato a 16 bit esteso a 32 bit, un offset e il numero di bit da leggere/scrivere. Dopodiché essendo che le istruzioni di load e store possono accedere in memoria ad un minimo di un byte e un massimo di quattro, la memoria è stata suddivisa su quattro bus identificati ciascuno da un bus enable ( $BE = 0,1,2,3$ ).

La funzione di load/store identificava il bus enable associato all'indirizzo al quale bisognava accederci basandosi solamente sull'offset, non tenendo conto anche del registro di partenza (Rs1), e di conseguenza si accedeva alla cella di memoria errata.

Per risolvere il bug abbiamo utilizzato un variabile che permette di calcolare il resto della divisione tra indirizzo di memoria di destinazione (Rs1) e quattro, cosicché le istruzioni load e store identificano il bus enable associato all'indirizzo in maniera corretta e di conseguenza di accedere alla giusta cella di memoria.

```
registers.a = registers.r[rs1];
registers.b = registers.r[rd];
registers.mar = signExtend(offset) + registers.a;
let rest = (registers.mar >>> 0) % 4 //permette di cap
let addr = Math.floor((registers.mar >>> 0) / 4) >>> 0;
registers.mdr = memory.load(addr, "IL");
registers.temp = rest
func(registers);
```

Figure 4: codice aggiunto

# Capitolo 4

## Nuove funzionalità

Il simulatore prevede un editor di testo, il quale permette di scrivere le istruzioni nel formato di codice assembly, che verranno successivamente eseguite dal processore. Inoltre, si può scegliere quale set di istruzioni utilizzare: DLX o RISC-V.

Durante l'esecuzione del codice si può simulare l'invio di un segnale interrupt alla CPU premendo il relativo bottone.

In basso troviamo lo spazio di indirizzamento del processore dove vengono mappate memorie come eprom, ram e altri dispositivi come led, counter. Si possono inoltre visualizzare le reti logiche dei vari dispositivi tramite una interfaccia personalizzata la quale è possibile configurarne le caratteristiche. Infine, si possono visualizzare nel dettaglio determinati indirizzi di memoria.

Sulla sinistra possiamo monitorare lo stato dei registri che vengono modificati durante l'esecuzione del codice.

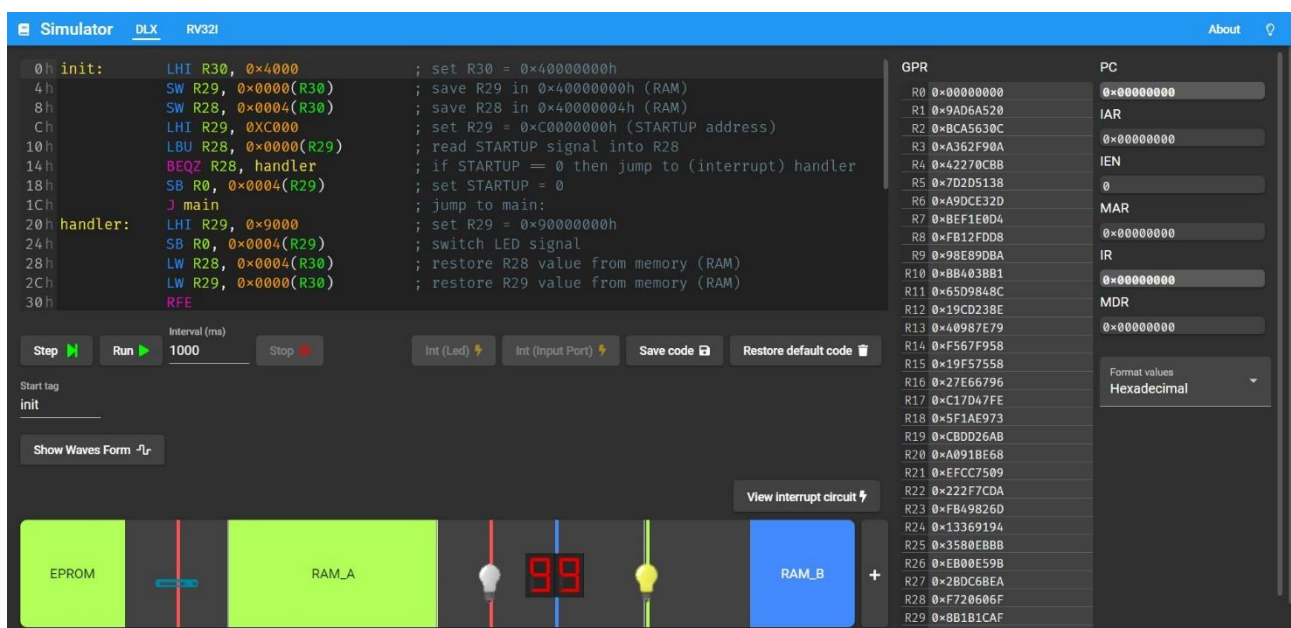


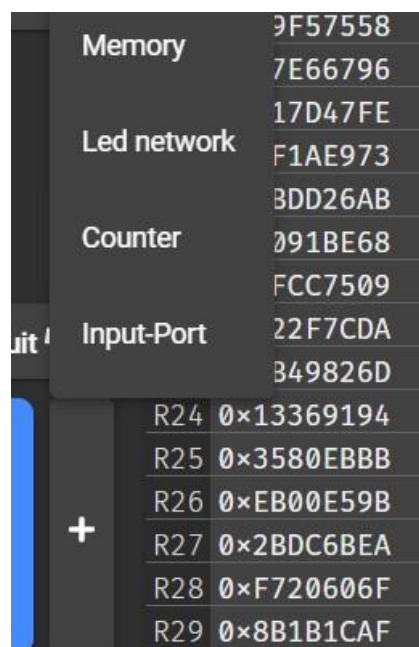
Figure 5: simulatore DLX

In questa tesi di laurea sono state implementate le seguenti funzionalità:

- L'aggiunta di una input port che simula la comunicazione e lo scambio di dati con l'esterno.
- L'evidenziazione dei registri al cambiare del loro valore.

## 4.1 Aggiunta Input-port

Nel simulatore vi è la possibilità di aggiungere allo spazio di indirizzamento un'interfaccia di input-port in grado di simulare il protocollo di handshake con un dispositivo esterno.



Memory	9F57558
	7E66796
	17D47FE
Led network	F1AE973
	3DD26AB
Counter	091BE68
	FCC7509
Input-Port	22F7CDA
	349826D
R24	0x13369194
R25	0x3580EBBB
R26	0xEB00E59B
R27	0x2BDC6BEA
R28	0xF720606F
R29	0x8B1B1CAF

Figure 6: aggiungere Input-Port

Cliccando sul tasto “+” collocato nella zona dei device è possibile mappare nello spazio d’indirizzamento un input-port cliccando su “*Input-Port*”, come è possibile visualizzare in Figura 6.



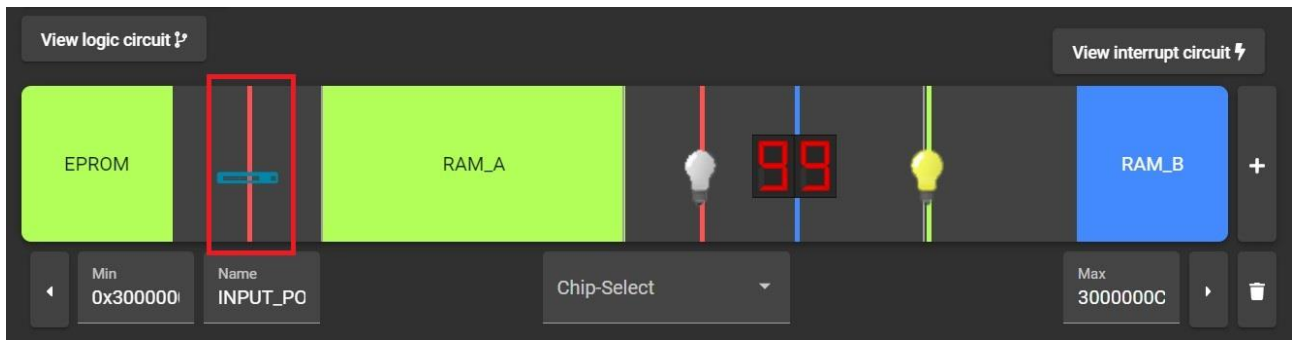


Figure 7: spazio indirizzamento con Input-Port

Come per le memorie, anche per gli altri dispositivi, è possibile scegliere gli indirizzi in cui mappare l'input-port e gli indirizzi in cui mappare i chip-select (Figura 7). Inoltre, selezionando la sua icona, è possibile visualizzare lo schema ai morsetti della rete logica cliccando su “*View logic circuit*” (Figura 7).

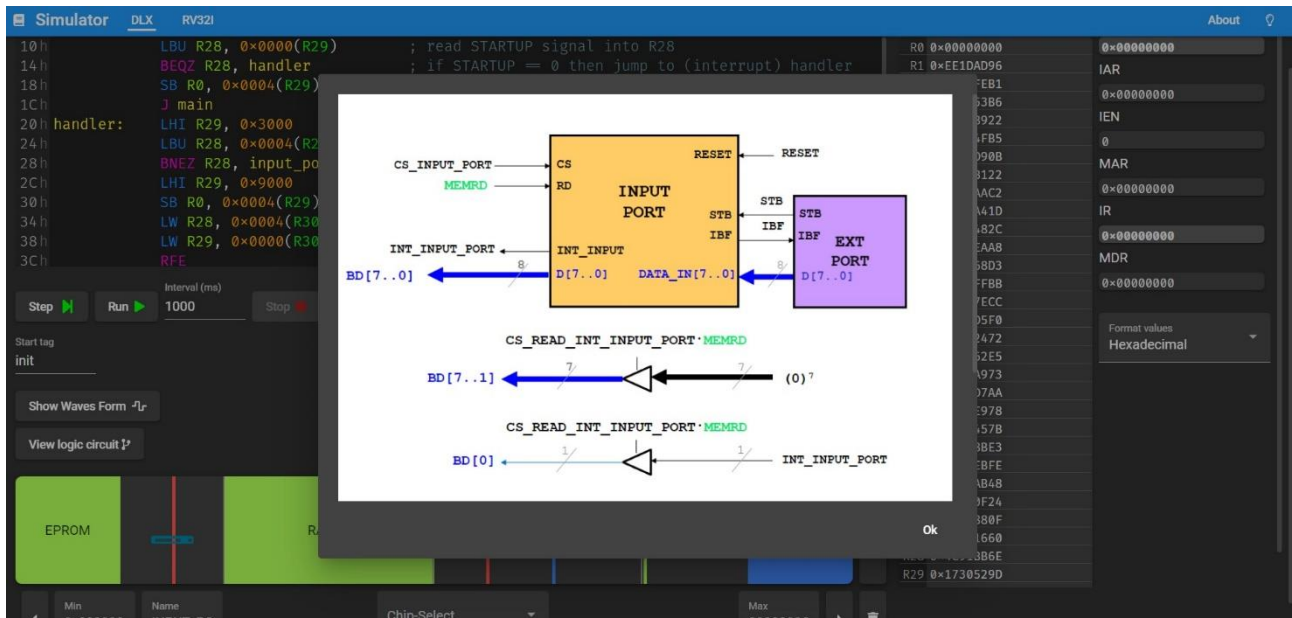


Figure 8: interfaccia di Input-Port

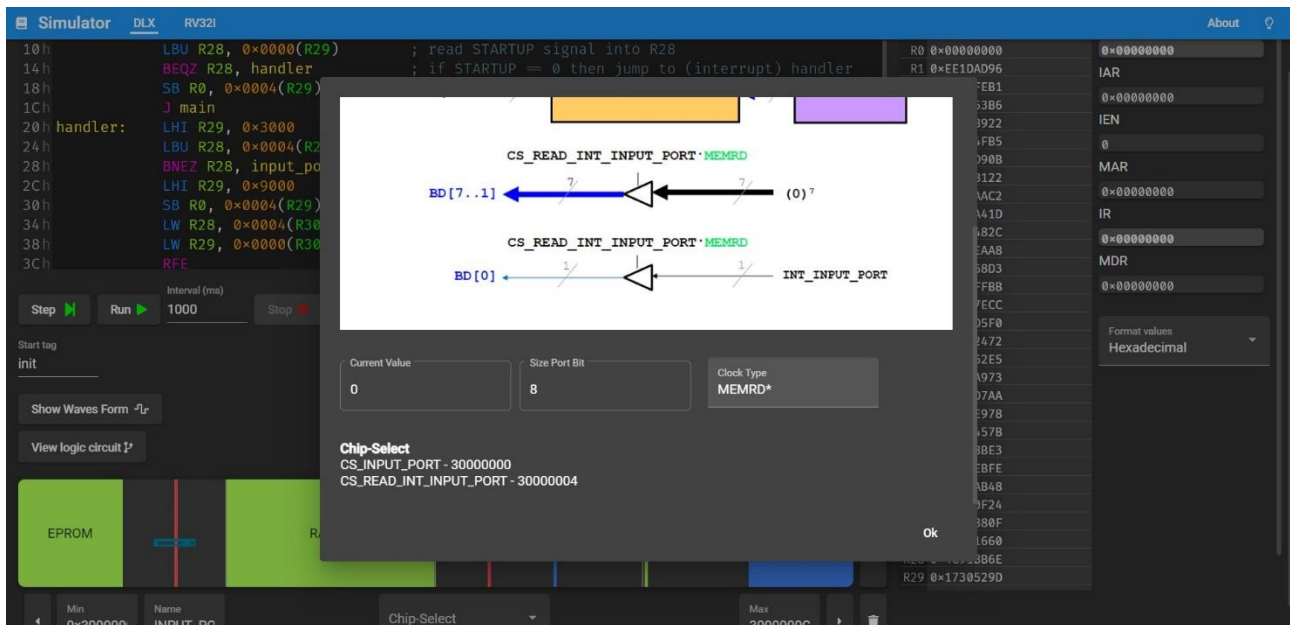


Figure 9: interfaccia Input-Port

Nella Figura 8 possiamo visualizzare la rete logica ai “morsetti” dell’input-port, e scorrendo verso il basso possiamo trovare tre campi:

- “*Current Value*” il quale indica il valore che ci viene fornito dall’Unità Esterna e che la CPU dovrà leggere.
- “*Size Port Bit*” indica la dimensione della porta in input, può essere di 8, 16, 32 bit, qualsiasi altra dimensione che l’utente andrà a scrivere genererà errore.
- “*Clock Type*” indica il tipo di clock che utilizza questa interfaccia, è settato a MEMRD\* che indica il fine ciclo di lettura, MEMWR\* non sarebbe corretto perché la cpu deve leggere il dato.

In fondo alla finestra troviamo i chip-select e i relativi indirizzi in cui sono mappati (Figura 9):

- *CS\_INPUT\_PORT*, effettuando una lettura a questo chip-select si ottiene il valore memorizzato nell’interfaccia di input.
- *CS\_READ\_INT\_INPUT\_PORT*, effettuando una lettura a questo chip-select possiamo vedere se l’interrupt relativo a questa porta è asserito, in caso positivo la CPU saprà che l’interruzione è stata scaturita da questo dispositivo.

Questo chip-select come vediamo in figura è in and logico con MEMRD perché il valore può essere ottenuto soltanto con un ciclo di lettura, un ciclo di scrittura non avrebbe senso.

Infine, premendo sul bottone “Int (Input-Port)” si aprirà un menu a tendina dove possiamo scegliere quale porta in input lancerà il segnale di interrupt. (Figura 10)

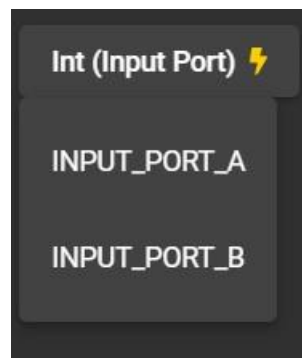


Figure 10: bottone interrupt Port

### 4.1.1 Esempi di funzionamento Input-port

Si osserva ora un esempio di funzionamento della input-port e di gestione dell'interrupt da parte del processore.

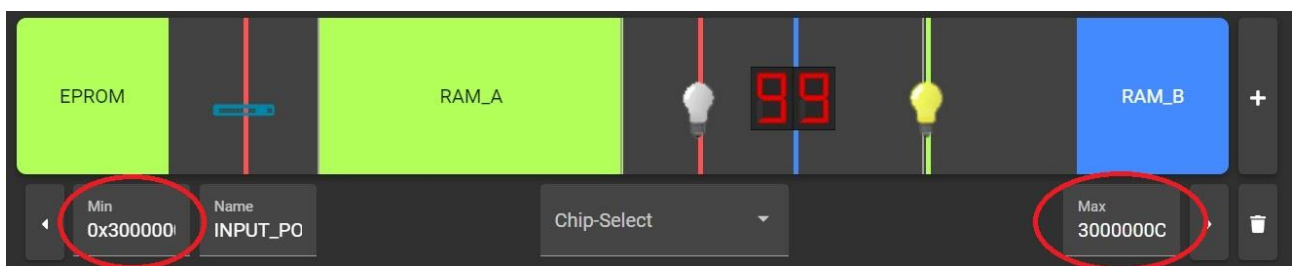


Figure 11: range indirizzi Input Port

La porta in input (Figura 11) è mappata in un range di indirizzi che va da 0x30000000 a 0x3000000C, mentre gli indirizzi dei chip-select sono rappresentati in Figura 9.

Generato l'interrupt premendo sul bottone in Figura 10, il processore farà il fetch dell'istruzione all'indirizzo 0h e andrà ad eseguire il codice *handler*.

```

20h handler:  LHI R29, 0x3000          ; set R29 = 0x30000000h (INPUT_PORT address)
24h          LBU R28, 0x0004(R29)   ; read interrupt INPUT_PORT signal into R28
28h          BNEZ R28, input_port    ; if INT_I = 0 then jump to (interrupt) input_port
2Ch          LHI R29, 0x9000          ; set R29 = 0x90000000h (LED address)
30h          SB R0, 0x0004(R29)       ; switch LED signal
34h          LW R28, 0x0004(R30)       ; restore R28 value from memory (RAM)
38h          LW R29, 0x0000(R30)       ; restore R29 value from memory (RAM)
3Ch          RFE

```

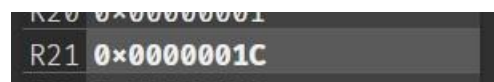
Nella prima riga del codice *handler* viene settato il registro R29 con l'indirizzo 30000000h. Nell'istruzione successiva viene effettuata una lettura all'indirizzo 30000004h dove è mappato CS\_READ\_INT\_INPUT\_PORT, ottenendo il valore dell'interrupt generato dalla porta. Se questo è diverso da zero effettuiamo un salto al codice di *input\_port*, altrimenti andremo ad eseguire altre istruzioni come sopra riportato.

```

74h input_port: LBU R20, 0x0000(R29) ; read data from input_port (CS_READ_PORT address)
78h          RFE
7Ch

```

Il codice *input\_port* andrà ad effettuare una load di un byte all'indirizzo 30000000h, dove è mappato CS\_INPUT\_PORT, ed otterremo il dato dell'input-port, salvandolo dunque nel registro R21.



R20	0x00000001
R21	0x0000001C

Di seguito possiamo vedere che il valore corrente salvato nella porta in input è cambiato.

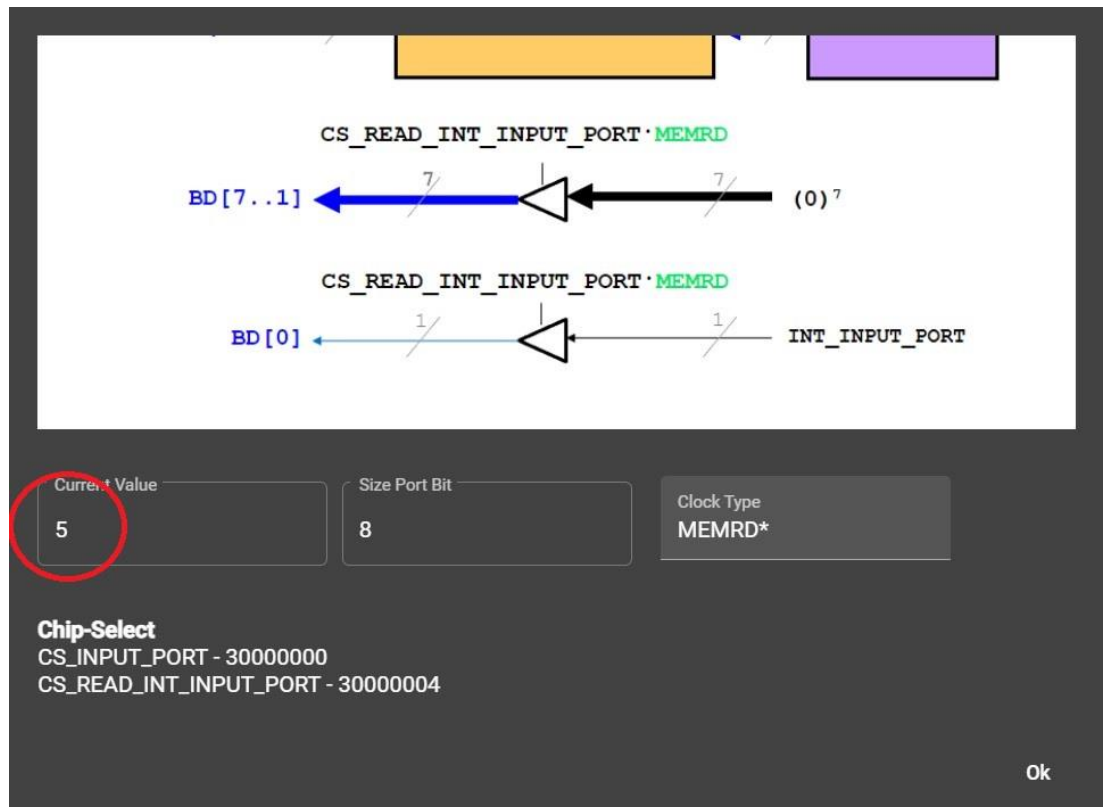


Figure 12: valore porta Input

## 4.2 Evidenziazione registri

Nel simulatore viene implementata la funzionalità che al cambio del valore dei registri prevede che questi vengono evidenziati in modo da renderli più immediati e leggibili. Per realizzare questa funzionalità abbiamo creato un array di lunghezza 31, dove ogni cella corrisponde ad un registro, contenente la regola di stile di default che troviamo nel simulatore (Figura 12).

```
this.rBold = ['normal', 'normal', 'normal', 'normal', 'normal', 'normal', 'normal', 'normal', 'normal', 'normal',
'normal', 'normal', 'normal', 'normal', 'normal', 'normal', 'normal', 'normal', 'normal',
'normal', 'normal', 'normal', 'normal', 'normal', 'normal', 'normal', 'normal', 'normal', 'normal', 'normal'];
```

Figure 13: array associato ai registri

Durante l'esecuzione del codice assembly da parte del simulatore, ogni volta che viene processata un'istruzione viene chiamata la funzione *setBold(registro)*, alla quale gli viene passato in input l'indice del registro da evidenziare, che andrà a modificare lo stile CSS della casella del registro stesso, e rimodificherà lo stile dei registri precedentemente modificati, eliminando l'evidenziazione.

```
public setBold(registroNum : number){  
    this.rBold[this.previous_register] = 'normal';  
    this.previous_register = registroNum;  
  
    if(registroNum > 0)  
        this.rBold[registroNum] = 'bold';  
}
```

Il risultato raggiunto è quello riportato di seguito.

R20	0x00000001
R21	0x0000001C
R22	0x55100100

# Capitolo 5

## Sviluppi futuri

Nell'ottica di possibili sviluppi futuri, si potrebbe realizzare un'interfaccia di output port, per poter simulare la comunicazione e lo scambio di dati con un dispositivo esterno lato output, cioè la CPU genererà un dato, e quando l'Unità Esterna sarà pronta, lo riceverà.

Inoltre, si potrebbe estendere i progetti delle porte che si interfacciano con l'esterno realizzando diagrammi che mostrino le forme d'onda dei segnali caratterizzati dal protocollo di handshake, per rendere ancora più chiaro la comunicazione del processore con i dispositivi esterni.

Infine, sarebbe utile affiancare all'applicazione un tool che permetta di creare delle reti logiche più complesse e articolate oltre a quelle già presenti nel simulatore.

# Conclusione

Le funzionalità di implementazione dell'interfaccia di input e di evidenziazione dei registri sono state realizzate seguendo la struttura progettuale scelta dai precedenti studenti che hanno lavorato al progetto del simulatore.

Si è cercato di progettare le implementazioni in maniera scalabile per favorire nuove estensioni future.

Numerose informazioni necessarie per lo sviluppo dell'elaborato sono state reperite dalle slide del corso Calcolatori Elettronici T tenuto dal professore Stefano

Mattocchia, ma è stata fondamentale la documentazione lasciata dai colleghi, in particolare Filippo Comastri che si è mostrato molto disponibile per poter effettuare il deploy del progetto sul server.

Inoltre, questo progetto mi ha dato la possibilità di approfondire le mie conoscenze tecniche imparando ad utilizzare il framework Angular e approfondire il linguaggio di Typescript.





# 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

[http://dlx-simulator.disi.unibo.it/simulator/assets/pdf/Tesi\\_Fabrizio\\_Maccagnani.pdf](http://dlx-simulator.disi.unibo.it/simulator/assets/pdf/Tesi_Fabrizio_Maccagnani.pdf)

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

[http://dlx-simulator.disi.unibo.it/simulator/assets/pdf/Tesi\\_Federico\\_Pomponii.pdf](http://dlx-simulator.disi.unibo.it/simulator/assets/pdf/Tesi_Federico_Pomponii.pdf)

[4] Filippo Comastri - “Estensione di un simulatore del processore DLX”

[http://dlx-simulator.disi.unibo.it/simulator/assets/pdf/Tesi\\_Filippo\\_Comastri.pdf](http://dlx-simulator.disi.unibo.it/simulator/assets/pdf/Tesi_Filippo_Comastri.pdf)

[5] Umberto Laghi - “Estensione del simulatore del processore DLX”

[http://dlx-simulator.disi.unibo.it/simulator/assets/pdf/Tesi\\_Umberto\\_Laghi.pdf](http://dlx-simulator.disi.unibo.it/simulator/assets/pdf/Tesi_Umberto_Laghi.pdf)

[6] Stefano Mattoccia - Slide corso Calcolatori Elettronici T Ingegneria Informatica: Linguaggio Macchina

[7] Stefano Mattoccia - Slide corso Calcolatori Elettronici T Ingegneria Informatica: Handshake

[8] elbuild.it - *Angular*

<https://www.elbuild.it/tecnologie/angular.html>

[9] geekandjob – *Cos'è typescript*

<https://www.geekandjob.com/wiki/typescript>

[10] mrw – *Introduzione ad Angular CLI*

[https://www.mrw.it/javascript/introduzione-angular-cli\\_12717.html](https://www.mrw.it/javascript/introduzione-angular-cli_12717.html)

[11] atlassian – *Cos'è GIT*

<https://www.atlassian.com/it/git/tutorials/what-is-git>