Architettura dei Calcolatori

Proposte Progettuali

Prof. Andrea Marongiu
Prof. Marko Bertogna
Dr. Gianluca Brilli

Lista progetti

- La composizione dei gruppi e la lista dei progetti assegnati sarà aggiornata sul link seguente:

  https://docs.google.com/spreadsheets/d/1JaRTs3d50DMc0XkPpqGzOBnHH2duLyr7iJsZpymJyFk/edit?usp=sharing

- Richiedere l'assegnamento del progetto tramite email.

Progetto 01 – Radix Sort

- Implementazione in assembly RISC-V dell'algoritmo di ordinamento **Radix Sort**. Un algoritmo di ordinamento non basato sul confronto degli elementi da ordinare. Prevede di ordinare gli elementi, partendo con l'analisi della cifra meno significativa e raggruppando gli elementi in base al valore di tale cifra.

```
RADIX-SORT (A, d)
    for i <- 1 to d
        do usa un ordinamento stabile per ordinare l'array A sulla cifra i
```

- Gruppo: *1 persona.*

Progetto 01 – Radix Sort

- Di seguito è riportato un esempio di funzionamento dell'algoritmo:

1° PASSO		2° PASSO		3° PASSO		4° PASSO		
253		10		5		5		5
346		253		10		10		10
1034	→	1034	→	127	→	1034	→	127
10		5		1034		127		253
5		346		346		253		346
127		127		253		346		1034

- Come algoritmo di ordinamento stabile, usare ad esempio **Counting Sort**.

- Gruppo: *1 persona.*

Progetto 02 – Merge Sort

- Implementazione in assembly RISC-V dell'algoritmo di ordinamento ricorsivo **Merge Sort**.

```
function mergesort (a[], left, right)
    if left < right then
        center ← (left + right) / 2
        mergesort(a, left, center)
        mergesort(a, center+1, right)
        merge(a, left, center, right)
```

- Gruppo: *1 persona.*

Progetto 03 – Quick Sort

- Implementazione in assembly RISC-V dell'algoritmo di ordinamento ricorsivo **Quick Sort**.

```
Procedure Quicksort(A)
Input A, vettore a1, a2, a3 .. an
  begin
    if n ≤ 1 then return A
    else
      begin
        scegli un elemento pivot ak
        calcola il vettore A1 dagli elementi ai di A tali che i ≠ K e ai ≤ ak
        calcola il vettore A2 dagli elementi aj di A tali che j ≠ K e aj > ak
        A1 ← Quicksort(A1)
        A2 ← Quicksort(A2)
        return A1 · (ak) · A2;
      end
```

- Gruppo: *1 persona.*

Progetto 04 – printf

- Realizzare un'implementazione in assembly RISC-V di una versione semplificata della funzione **printf** del C, basata sulla write vista in classe.
- Esempio:

  pr*intf(“You are %d years old\n”, 15);*

- Che in assembly sarebbe ad esempio:

```
.section .data
        str: .space 50
.section .text
_start:
        # init printf registers
        ...
        la    a0, str
        jal   ra, printf
```

- Gruppo: *1 persona.*

Progetto 05 – Sparse Matrix Multiplication

- Realizzare in assembly RISC-V una variante del prodotto matriciale visto a lezione. L'implementazione deve tenere conto della struttura delle matrici da moltiplicare, nello specifico fornire un'implementazione per **matrici sparse**, ovvero matrici che contengono un elevato numero di zeri al loro interno.

- Gruppo: 2 *persone.*

Progetto 05 – Sparse Matrix Multiplication

- Utilizzare ad esempio il formato *Yale Sparse Matrix Format*:

For example, the matrix

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 5 & 8 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 6 & 0 & 0 \end{pmatrix}$$

is a 4×4 matrix with 4 nonzero elements, hence

```
A  = [ 5 8 3 6 ]
IA = [ 0 0 2 3 4 ]
JA = [ 0 1 2 1 ]
```

- A: elementi != 0;
- IA: numero di elementi != 0 nella riga i - 1 della matrice.
- JA: indice di colonna dell'elemento A(i).

https://en.wikipedia.org/wiki/Sparse_matrix

- Gruppo: 2 *persone.*

Progetto 06 – Ricerca Binaria

- Implementare in assembly RISC-V un algoritmo di ricerca binaria su un vettore.
- Partire implementando un algoritmo di ricerca sequenziale.
- Dopodiché confrontare l'implementazione con una versione binaria, operante su un vettore ordinato.

- Gruppo: *1 persona.*

Progetto 06 – Ricerca Binaria

- Di seguito si riporta un possibile pseudocodice che risolve il problema della ricerca binaria:

```
// initially called with low = 0, high = N-1
BinarySearch(A[0..N-1], value, low, high) {
    // invariants: value > A[i] for all i < low
                   value < A[i] for all i > high
    if (high < low)
        return not_found // value would be inserted at index "low"
    mid = (low + high) / 2
    if (A[mid] > value)
        return BinarySearch(A, value, low, mid-1)
    else if (A[mid] < value)
        return BinarySearch(A, value, mid+1, high)
    else
        return mid
}
```

- Gruppo: *1 persona.*

Progetto 07 – FIR Filter

- Un filtro è un sistema che realizza una funzione di trasformazione di un segnale di ingresso x(n). L'operazione di filtraggio può essere quella di eliminare alcune bande di frequenza.
- Possibile formalizzazione di un FIR:

$$y[n] = b_0 x[n] + b_1 x[n-1] + \cdots + b_N x[n-N]$$
$$= \sum_{i=0}^{N} b_i \cdot x[n-i],$$

- Gruppo: *1 persona.*

Progetto 07 – FIR Filter

- Implementare in assembly RISC-V un filtro *Finite Impulse Response* (FIR).
- Fare riferimento ad esempio alle seguenti trattazioni:

https://sestevenson.wordpress.com/implementation-of-fir-filtering-in-c-part-1/

http://www.grix.it/UserFiles/theremino/File/Semplice%20implementazione%20dei%20filtri%20FIR.pdf

- Gruppo: *1 persona.*

Progetto 08 – Trasformata DCT

- La **Trasformata Discreta del Coseno** (DCT) è un'operazione spesso usata nell'image processing, essa è in grado di rilevare le variazioni di informazione tra un'area e quella contigua di un'immagine digitale trascurando le ripetizioni.
- Di seguito una possibile formalizzazione:

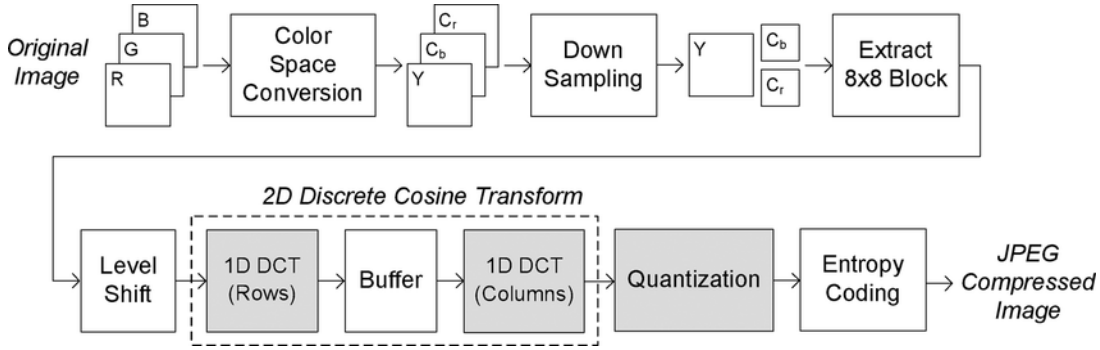
$$D(i,j) = \frac{1}{\sqrt{2N}} C(i)C(j) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} p(x,y) \cos\left[\frac{(2x+1)i\pi}{2N} \right] \cos\left[\frac{(2y+1)j\pi}{2N} \right]$$

$$C(u) = \left\{ \begin{array}{ll} \frac{1}{\sqrt{2}} & \text{if } u = 0 \\ 1 & \text{if } u > 0 \end{array} \right\}$$

- Gruppo: *1 persona.*

Progetto 08 – Trasformata DCT

- L’utilizzo principale della Trasformata DCT è come blocco di compressione nel formato JPEG:



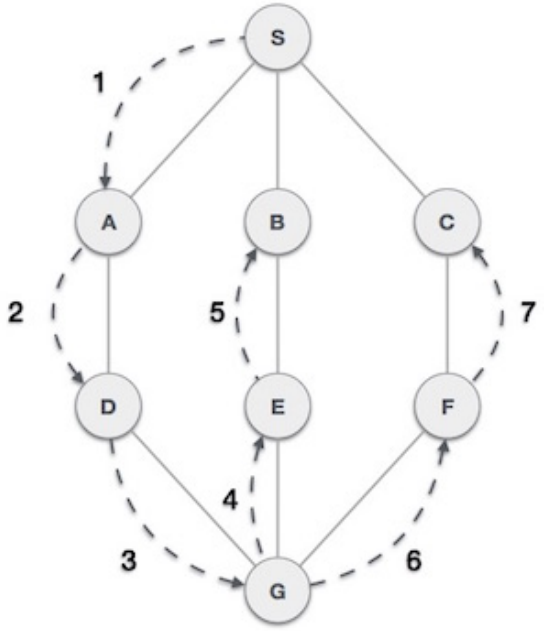
- Fornire un’implementazione assembly RISC-V della DCT, eventualmente prendendo spunto dalla seguente pagina web: https://www.geeksforgeeks.org/discrete-cosine-transform-algorithm-program/

- Gruppo: *1 persona.*

Progetto 09 – Depth First Search

- Implementare in assembly RISC-V l'algoritmo su grafo di visita in profondità (DFS).
- Per l'implementazione dell' algoritmo prendere spunto dal seguente esempio:

https://www.tutorialspoint.com/data_structures_algorithms/depth_first_traversal.htm

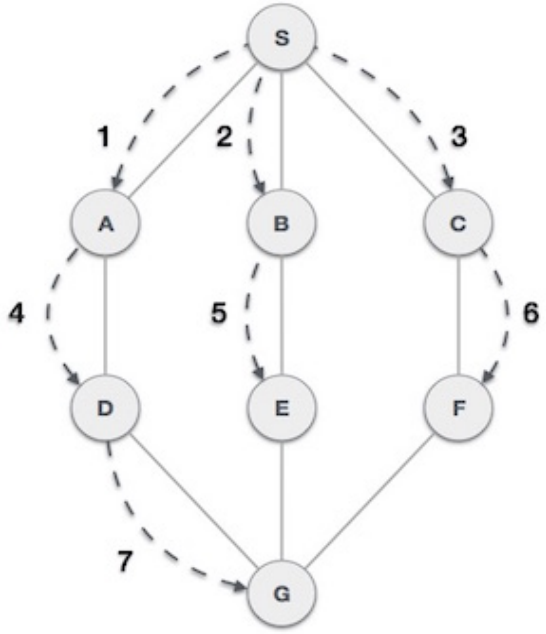


- Gruppo: *1 persona.*

Progetto 10 – Breadth First Search

- Implementare in assembly RISC-V l'algoritmo su grafo di visita in ampiezza (BFS).
- Per l'implementazione dell' algoritmo prendere spunto dal seguente esempio:

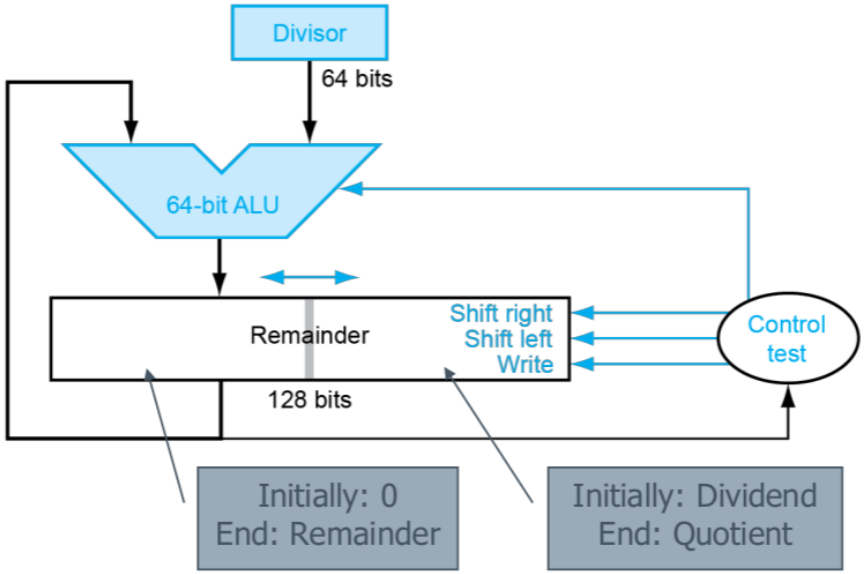
https://www.tutorialspoint.com/data_structures_algorithms/depth_first_traversal.htm



- Gruppo: *1 persona.*

Progetto 11 – Circuito Divisore

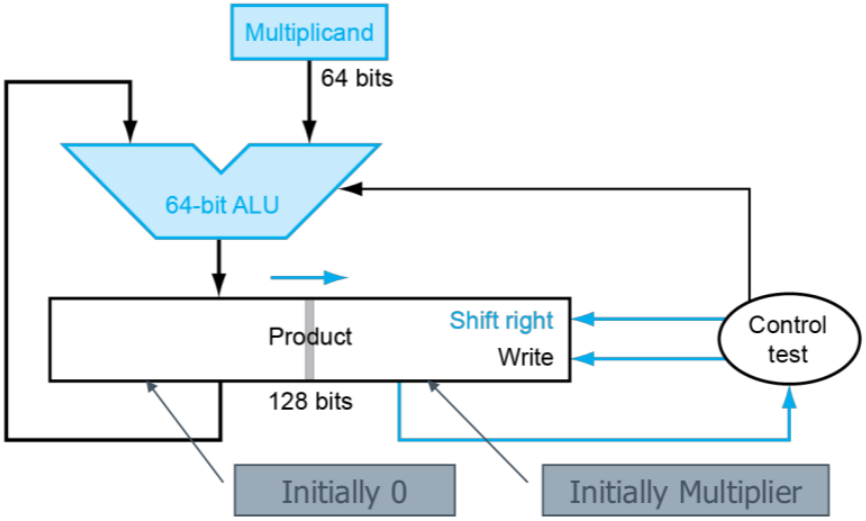
- Utilizzare il tool Logisim per realizzare un circuito divisore (come visto a lezione), utilizzando i blocchi fondamentali realizzati durante le esercitazioni.



- <u>Gruppo</u>: 2 *persone.*

Progetto 12 – Circuito Moltiplicatore

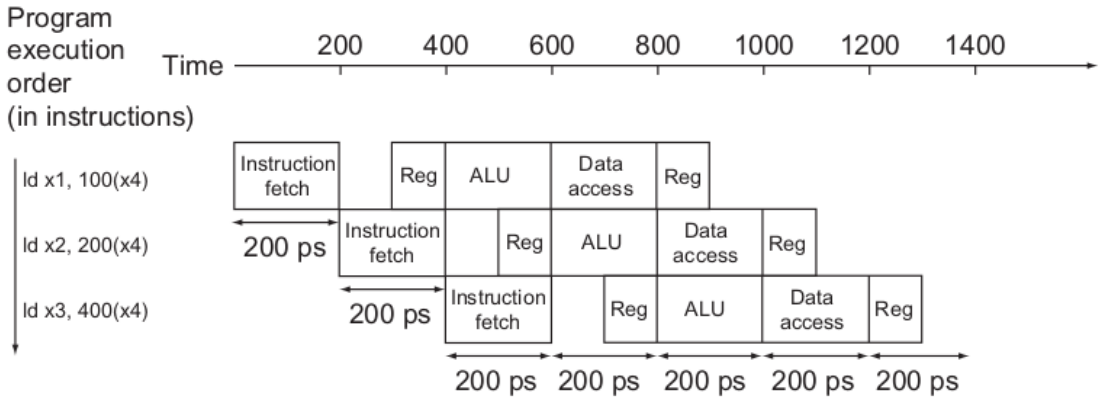
- Utilizzare il tool Logisim per realizzare un circuito moltiplicatore (come visto a lezione), utilizzando i blocchi fondamentali realizzati durante le esercitazioni.



- Gruppo: 2 *persone.*

Progetto 13 – Versione Pipelined di RISC-V

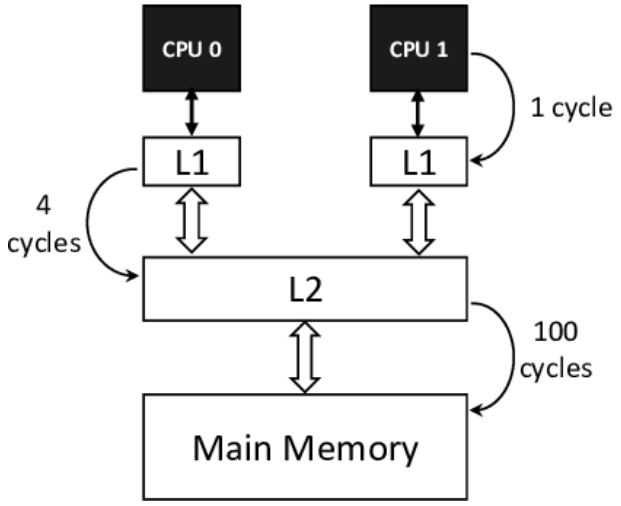
- Implementare in Logisim un sottoinsieme di RISC-V in **versione pipelined**, come visto a lezione. Partire dal processore di esempio che si è costruito durante le esercitazioni.



- Gruppo: 3 *persone.*

Progetto 14 – Realizzazione di un Sistema di Cache

- Implementare in Logisim una piccola gerarchia di memoria, nella quale è presente un livello di cache, tra i registri del processore e la memoria.
- Il sistema deve essere integrato nell'architettura di esempio realizzata durante le esercitazioni.



- Gruppo: 3 *persone.*

Progetto 15 – RISC-V Dual Issue

- Durante il corso si è vista una versione basilare di RISC-V, il quale è in grado di svolgere un'istruzione per ciclo di clock.
- Prototipare tramite Logisim, una versione Multiple-Issue di RISC-V, in grado di eseguire più istruzioni nello stesso ciclo di clock.

	ALU or branch instruction	Data transfer instruction	Clock cycle
Loop:		ld x31, 0(x20)	1
	addi x20, x20, -8		2
	add x31, x31, x21		3
	blt x22, x20, Loop	sd x31, 8(x20)	4

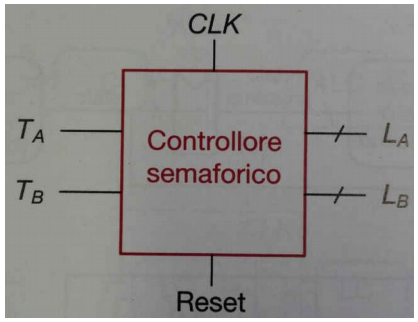
- Gruppo: 3 *persone.*

Progetto 16 – Ascensore

- Realizzare un controllore digitale per un ascensore di un edificio a 25 piani. Il controllore si deve interfacciare con un certo numero di pulsanti che permettono di selezionare il piano a cui andare e genera in uscita l'indicazione del piano al quale si trova l'ascensore.
- Progettare l'FSM (di Mealy o Moore), specificare la codifica degli stati e sintetizzare tramite mappe di Karnaugh, successivamente realizzare il circuito in Logisim.
- Gruppo: *1 persona.*

Progetto 17 – Controllore Semaforico

- Realizzare un controllore semaforico, posto al centro di un incrocio stradale. All'interno dell'incrocio sono presenti due sensori di traffico (*Ta* e *Tb*), tali sensori indicano "0" o "1" rispettivamente se rilevano traffico o meno. Progettare una macchina a stati come in figura, avente i sensori, *CLK* e *Reset* come input e lo stato dei semafori come output.
- Progettare l'FSM, specificare la codifica degli stati, sintetizzare con mappe di Karnaugh e realizzare il circuito in Logisim.
- Gruppo: *1 persona.*



Progetto 18 – Riconoscitore di Sequenze

- Realizzare un circuito digitale in grado di riconoscere le sequenze binarie A = “1101” o B = “1110”, successivamente implementare due *contatori*, per ogni sequenza e collegarli a due *display a 7 segmenti*, in modo tale da avere un conteggio di quante sequenze sono state riconosciute (*es: 4 sequenze A e 5 sequenze B*).
- Progettare l’FSM (di Mealy o Moore), specificare la codifica degli stati e sintetizzare tramite mappe di Karnaugh, successivamente realizzare il circuito in Logisim.
- Gruppo: *1 persona.*

Progetto 19 – Distributore Automatico

- Realizzare un circuito digitale in grado di gestire un distributore automatico di bottigliette. Ogni bottiglietta costa *50* centesimi. Il distributore accetta monete da *5, 10, 20 e 50* centesimi: quando il totale è sufficiente eroga la bottiglietta e restituisce l'eventuale resto. Progettare una FSM dotata di un ingresso per moneta e le seguenti uscite: *Eroga, Rendi5, Rendi10, Rendi20 e Rendi50*. Quando l'importo raggiunge o supera *50* si deve attivare *Eroga* ed eventualmente *Rendi*.
- Sequire il workflow visto a lezione e testare il circuito in Logisim.
- Gruppo: *1 persona.*

Progetto 20 – Contatore a Codice Gray

- Progettare un contatore modulo 8 a codice Gray, senza ingressi e con tre uscite (si ricordi che un contatore modulo N conta da 0 a N-1, quindi riparte da 0). Il contatore parte da 000 e generare in uscita il prossimo codice Gray a ogni fronte di salita del clock.
- Progettare la FSM relativa, specificare la codifica degli stati, sintetizzare il sistema tramite mappe di Karnaugh e testare il circuito in Logisim.
- Gruppo: *1 persona.*

Decimal	b_2	b_1	b_0	g_2	g_1	g_0
0	0	0	0	0	0	0
1	0	0	1	0	0	1
2	0	1	0	0	1	1
3	0	1	1	0	1	0
4	1	0	0	1	1	0
5	1	0	1	1	1	1
6	1	1	0	1	0	1
7	1	1	1	1	0	0