UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

Dipartimento di Scienze Fisiche,
Informatiche e Matematiche

Esercitazione 05:
CPU design
Architettura dei calcolatori [MN1-1143]

Corso di Laurea in Ing. Informatica
(D.M.270/04) [16-215]
Anno accademico 2020/2021

Dott. Gianluca Brilli
gianluca.brilli@unimore.it
Prof. Marko Bertogna
Marko.bertogna@unimore.it

CPU Design
Instruction Set

- *In questa esercitazione andremo a progettare una semplice architettura di esempio il cui instruction set (ISA) è composto dalle seguenti istruzioni:*

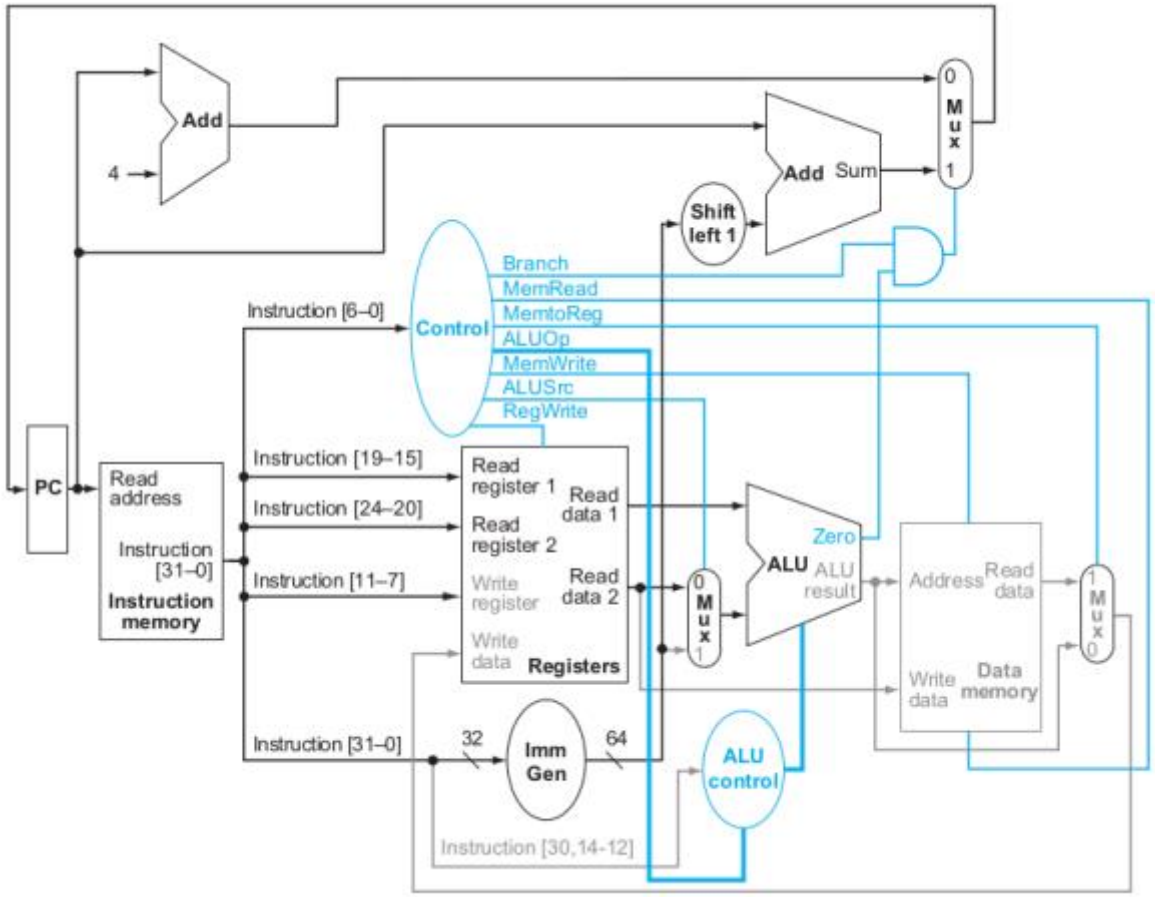
- *Istruzioni di Memoria: operanti su byte, in particolare load/store byte (lb e sb);*

- *Istruzioni ALU: and, or, addizione e sottrazione (and, or, add, sub);*

- *Istruzioni di Salto: per il controllo del flusso del programma, nello specifico, branch if equal (beq)*

CPU Design

Datapath



CPU Design
Codifica delle istruzioni

➔ *Basate sulle specifiche di codifica del RISC-V:*

Formato	Istruzione	Opcode	funct3	funct7	ALU Action	ALU Control
R-Type	add	10	00	00	add	10
	sub	10	00	01	sub	11
	and	10	11	00	and	00
	or	10	10	00	or	01
SB-Type	beq	01	xx	xx	sub	11
I-Type	lb	00	xx	xx	add	10
S-Type	sb	11	xx	xx	add	10

CPU Design
Codifica delle istruzioni

- *Manteniamo uno standard simile alla codifica delle istruzioni vista in classe per l'architettura RISC-V a 64 bit. Nel nostro caso riduciamo la codifica a soli 16 bit di istruzioni.*

- *Istruzioni I-Type:*

15		*…*		*0*
Imm[7:0]	**rs1**	**funct3**	**rd**	**op**
8bit	*2bit*	*2bit*	*2bit*	*2bit*

CPU Design

Codifica delle istruzioni

➔ *Istruzioni S-Type:*

15		…			0
Imm[7:2]	**rs2**	**rs1**	**funct3**	**Imm[1:0]**	**op**
6bit	*2bit*	*2bit*	*2bit*	*2bit*	*2bit*

➔ *Istruzioni SB-Type:*

15		…			0
Imm[0, 7, 5:2]	**rs2**	**rs1**	**funct3**	**Imm[1, 6]**	**op**
6bit	*2bit*	*2bit*	*2bit*	*2bit*	*2bit*

➔ *Istruzioni R-Type:*

15		…				0
xxxx	**funct7**	**rs2**	**rs1**	**funct3**	**rd**	**op**
4bit	*2bit*	*2bit*	*2bit*	*2bit*	*2bit*	*2bit*

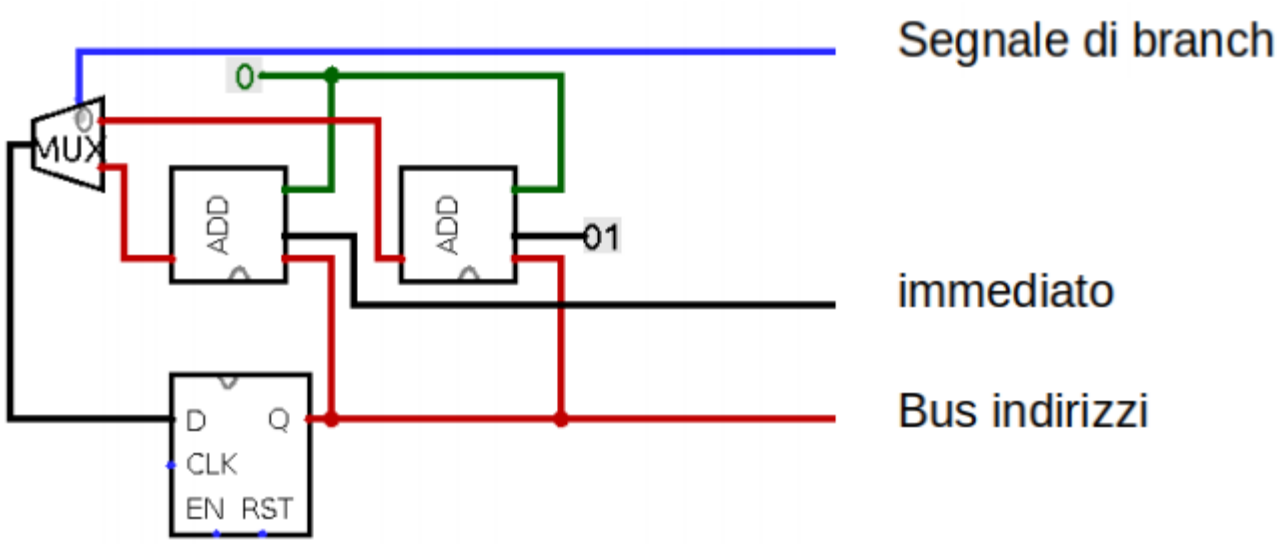
Program Counter

Esercizio 01

- *In questa implementazione prevediamo l'utilizzo di due differenti zone di memoria: la instruction memory e la data memory.*
- *Realizzare secondo il precedente schema, la circuiteria di gestione del program counter (PC), in modo tale che venga incrementato ad ogni ciclo di clock;*
- *Inserire inoltre la possibilità di incrementi maggiori di 1 (branch).*

Program Counter

Esercizio 01 - Soluzione



- ➔ *l'adder a sinistra somma l'uscita del PC con gli immediati (beq x1, x2, imm)*
- ➔ *l'adder a destra gestisce il normale incremento del PC ad ogni ciclo di clock.*

Connessione Regfile/ALU

Esercizio 02

- ***Requisiti da soddisfare**:*
- ***1)** Prevedere la possibilità di sommare il valore di un registro con un immediato e di andare in memoria all'indirizzo calcolato dall'ALU.*
- *Esempio:*
  - *sb X1, **4(X3)***
    - *addr = X3 + 4*
    - *mem[addr]*

Secondo operando, necessario sommare un offset e andare in memoria. Connettere il risultato della somma al bus indirizzi.

Connessione Regfile/ALU

Esercizio 02

- ***Requisiti da soddisfare**:*
- ***2)** Prevedere la possibilità di scrivere in memoria il valore contenuto all'interno di un registro.*
- *Esempio:*

  - *sb **X1**, 4(X3)* → *mem[addr] = X1*

Dopo aver calcolato l'indirizzo su cui andare a scrivere, è necessario scrivere il valore contenuto nel registro. Connettere il primo registro al bus dati.

Connessione Regfile/ALU

Esercizio 02

- ***Requisiti da soddisfare**:*
- ***3)** Prevedere la possibilità di scrivere in un registro il risultato di un'operazione ALU.*
- *Esempio:*
  - *add **X1**, X2, X3*

Dopo l'operazione ALU, viene scritto il valore della somma all'interno del Regfile.

Connessione Regfile/ALU

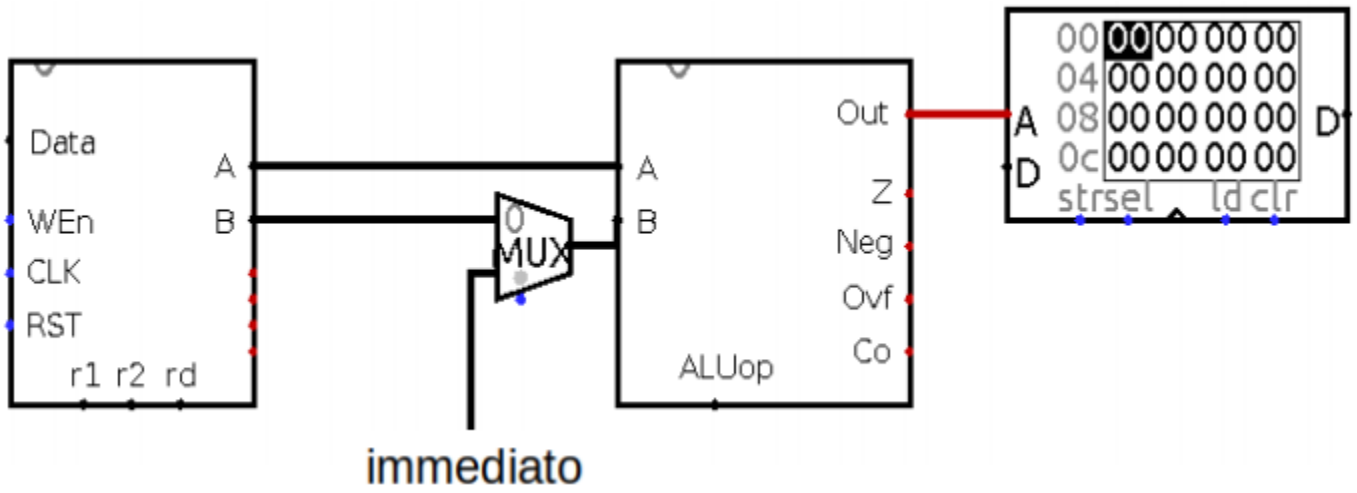
Esercizio 02

- ***Requisiti da soddisfare**:*
- ***4)** Prevedere la possibilità di leggere un byte dalla memoria e caricarlo in un registro.*
- *Esempio:*
  - *lb **X1**, 0(X3)*

Dopo il calcolo dell'indirizzo (analogo alla store vista prima), viene letto un dato dalla memoria e scritto all'interno del Regfile.

Connessione Regfile/ALU

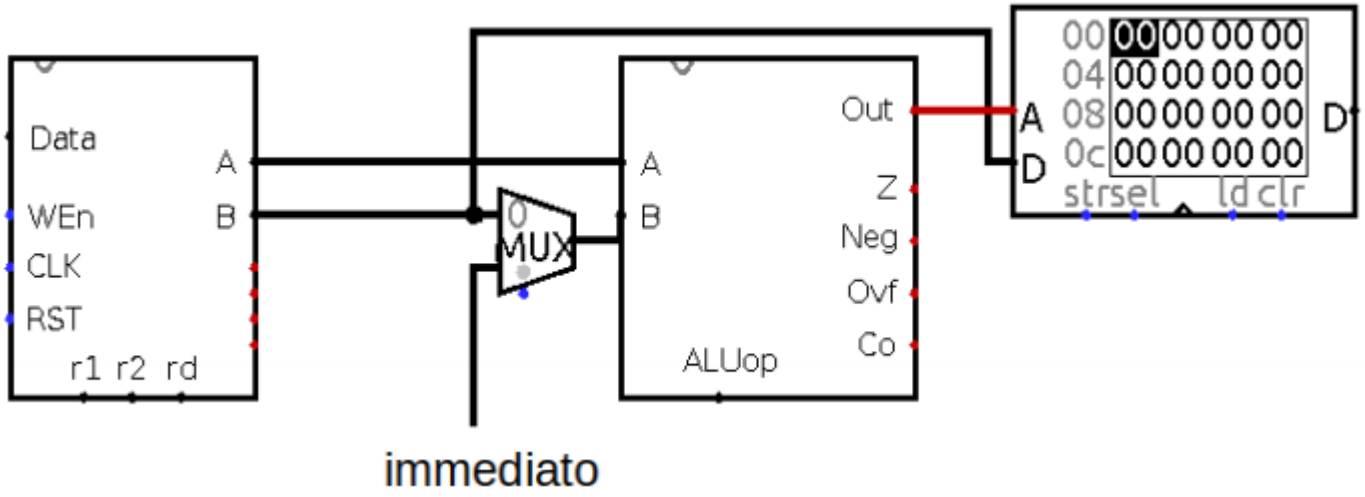
Esercizio 02 - Circuito



➔ ***1)*** *Somma tra immediato (operando B) e valore contenuto in un registro (operando A) e accesso in memoria all'indirizzo calcolato dall'ALU.*

Connessione Regfile/ALU

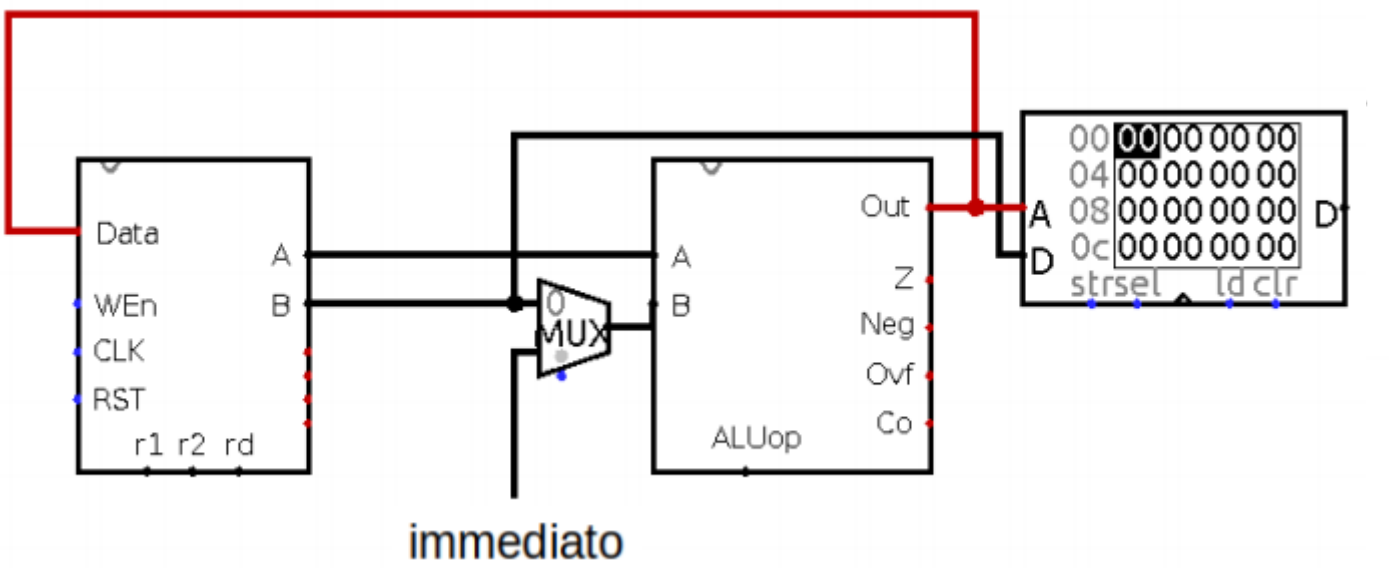
Esercizio 02 - Circuito



➔ ***2)*** *Scrittura in memoria del contenuto di un registro.*

Connessione Regfile/ALU

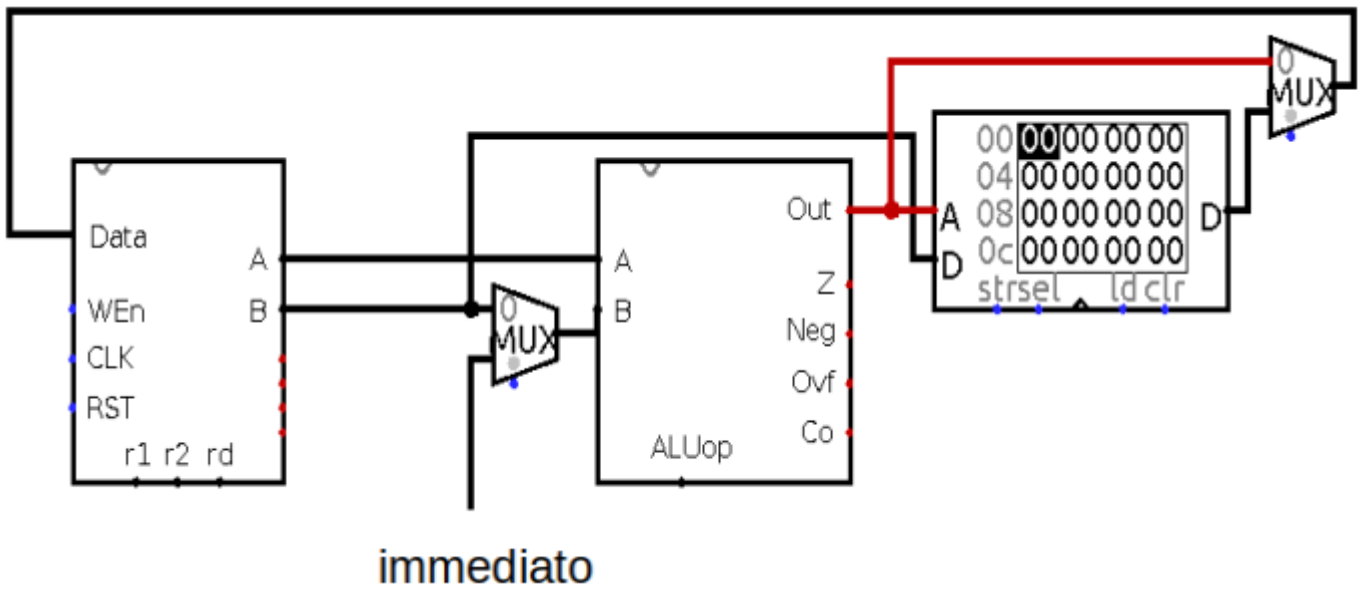
Esercizio 02 - Circuito



➔ ***3)*** *Scrittura all'interno del Regfile del risultato di un'operazione ALU.*

Connessione Regfile/ALU

Esercizio 02 - Circuito



➔ ***4)*** *Lettura di un byte dalla memoria e scrittura all'interno del Regfile.*

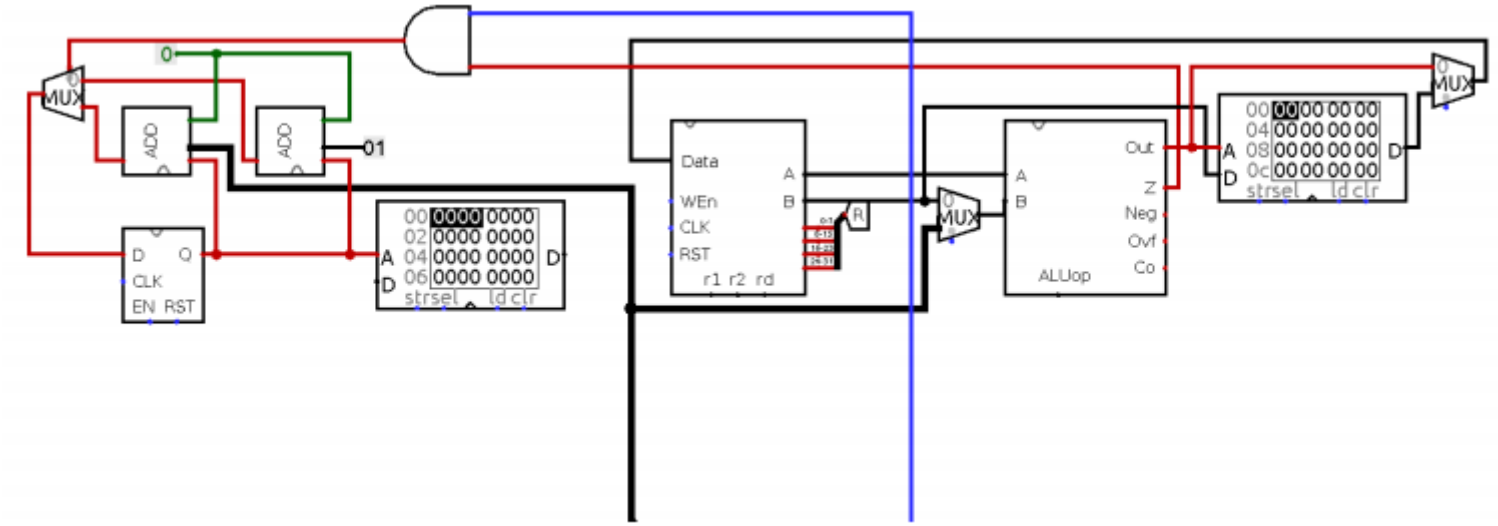
Connessione PC/Regfile/ALU

Esercizio 03

- *Completare il design del datapath della CPU, andando ad interconnettere tra di loro ciò che è stato realizzato nei precedenti esempi precedenti:*
  - *Program Counter;*
  - *Register File;*
  - *ALU;*
  - *Memoria Dati / Istruzioni.*

Connessione PC/Regfile/ALU

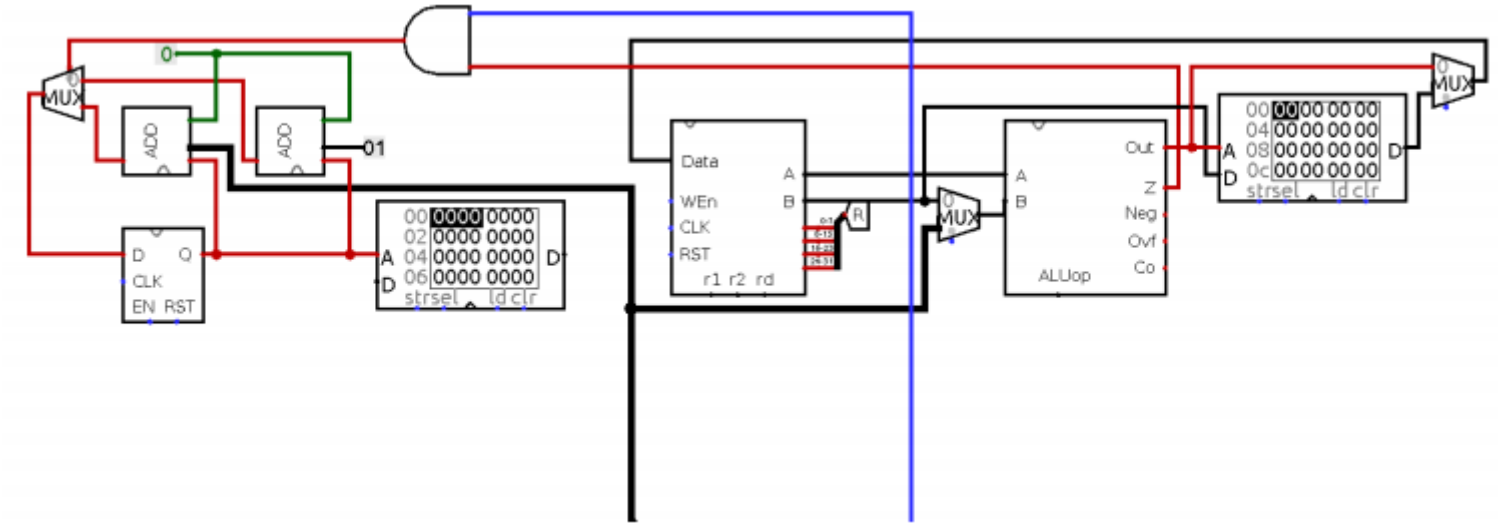
Esercizio 03 - Circuito



➔ *Il filo blu e nero (scuro) sono due segnali provenienti dalla control unit, che al momento non abbiamo ancora, quindi possiamo tenerli scollegati.*

Connessione PC/Regfile/ALU

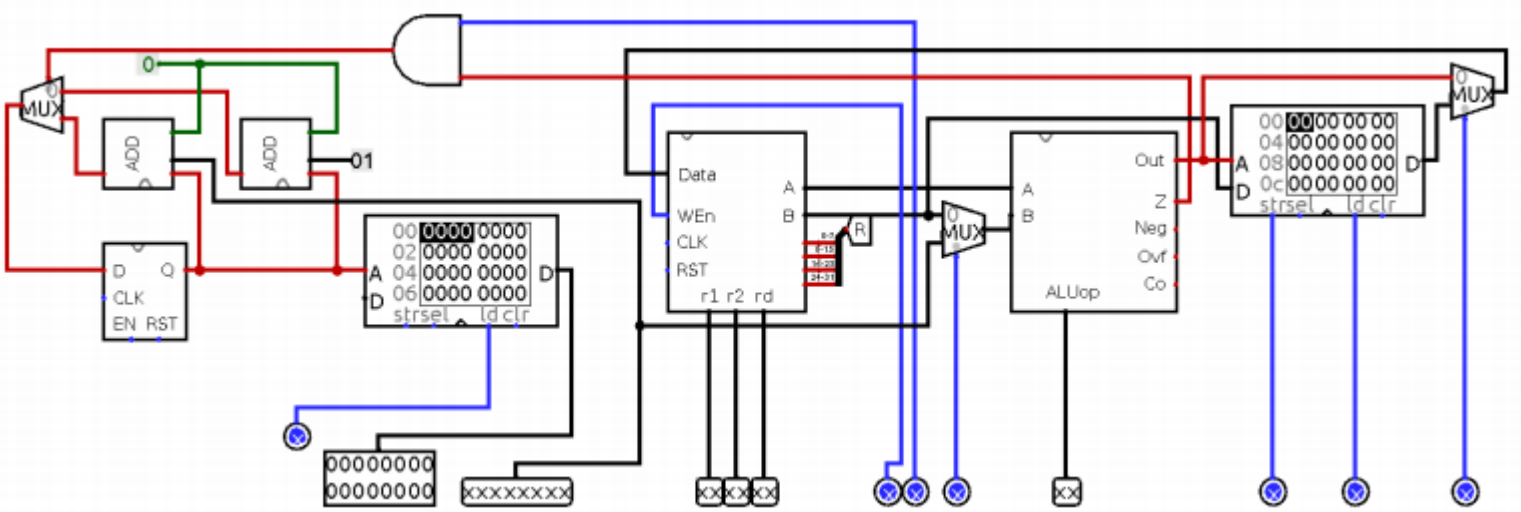
Esercizio 03 - Circuito



- ➔ *Nello specifico abbiamo interconnesso il bus degli immediati ed il flag zero dell'ALU al segnale di controllo dei jump.*
- ➔ *L'and abilita il jump solo se impostato dalla control unit.*

Control Unit

Esercizio x casa



➔ *Progettiamo un'unità di controllo che in base all'istruzione da effettuare vada a settare in maniera automatica tutti i segnali di controllo di tutti i moduli della CPU (i pin di ingresso / uscita riportati in basso in figura).*

Control Unit

Esercizio x casa

➔ *Realizzare un circuito digitale che vada ad implementare le seguenti tabelle di verità:*

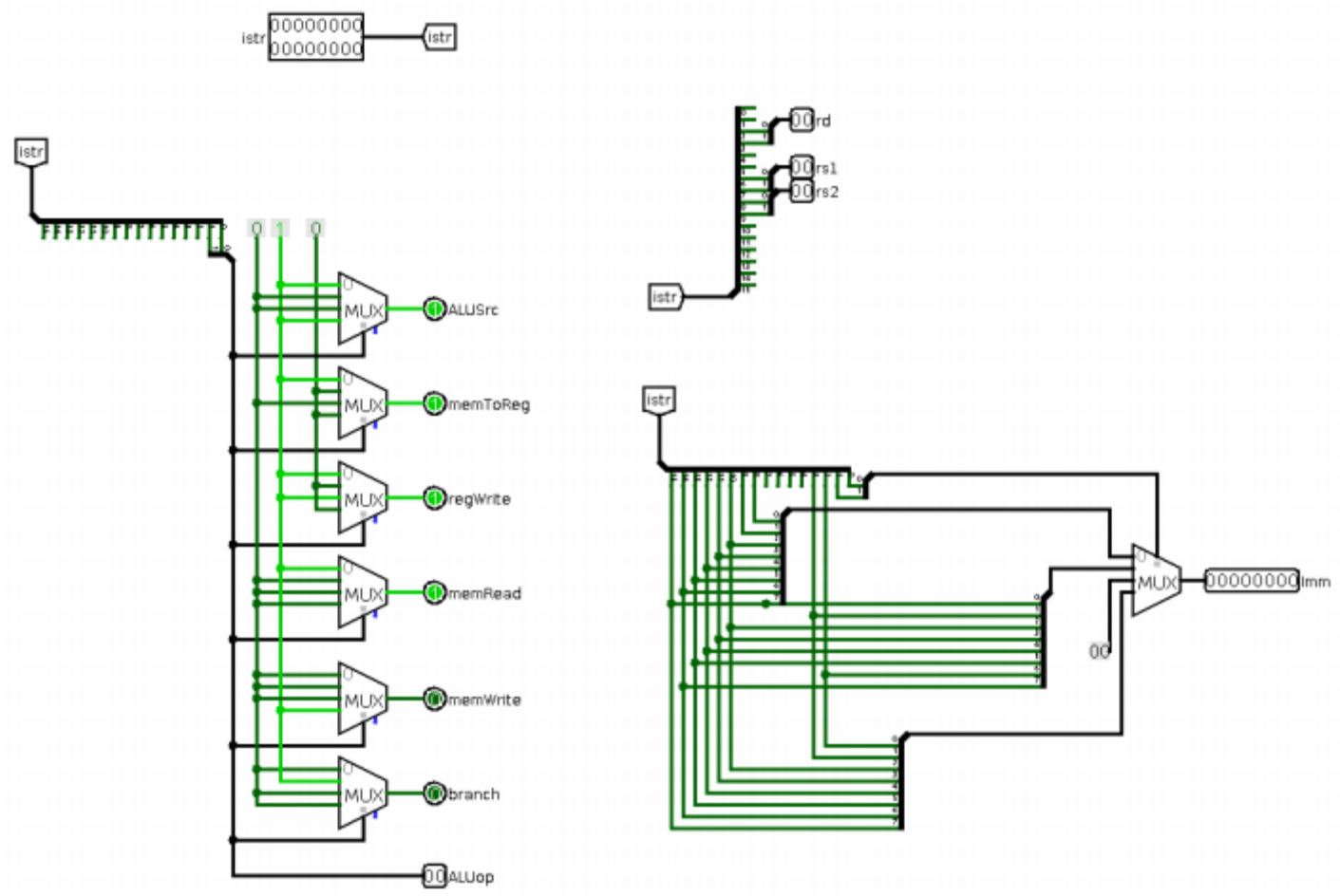
Istruzione	ALUsrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUop1	ALUop0
R-Type	0	0	1	0	0	0	1	0
lb	1	1	1	1	0	0	0	0
sb	1	x	0	0	1	0	1	1
beq	0	x	0	0	0	1	0	1

➔ *Per quanto riguarda le istruzioni R-Type:*

opcode	funct7	funct3	ALU Control	Istruzione
10	00	00	10	add
10	01	00	11	sub
10	00	11	00	and
10	00	10	01	or

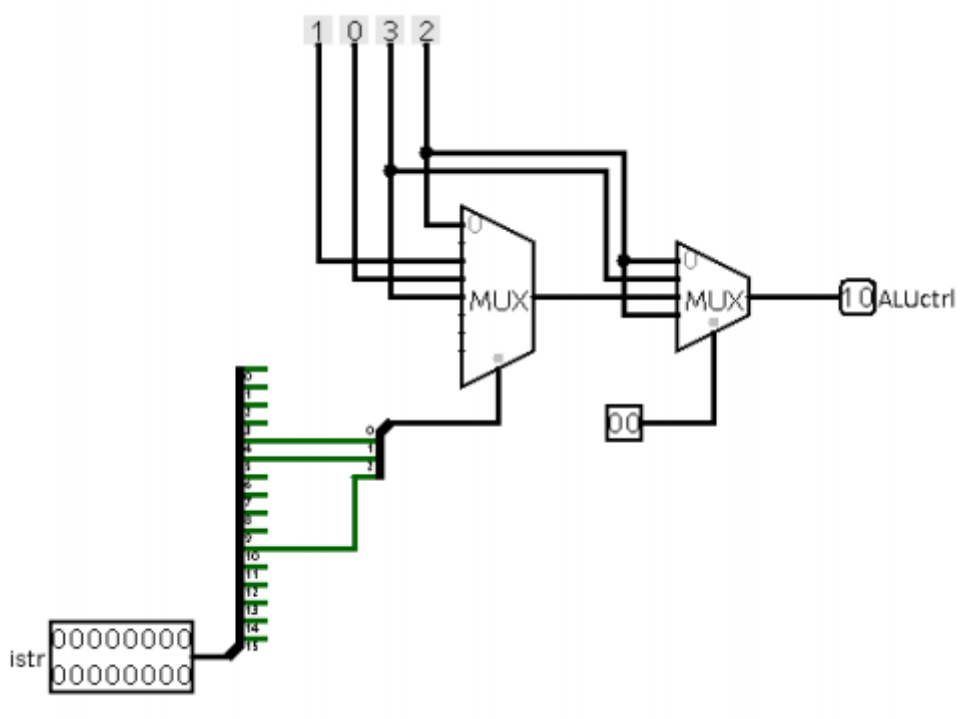
Control Unit principale

Esercizio x casa



Control Unit ALU

Esercizio x casa



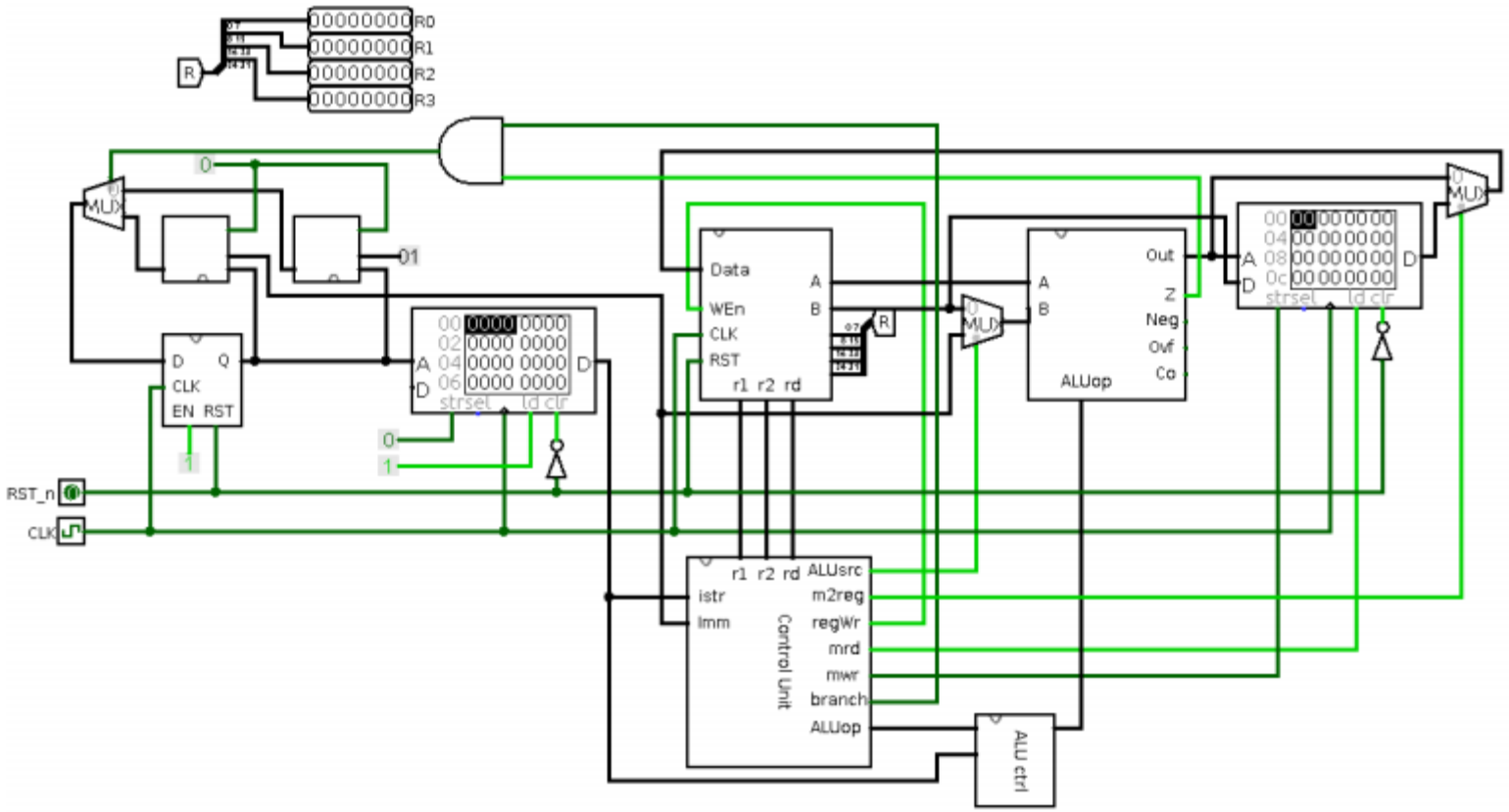
CPU Datapath

Esercizio 04

- *Partire dal circuito realizzato nella parte 04, importare la control unit realizzata al passo precedente.*
- *Successivamente sostituire tutti i pin nei segnali di controllo con le uscite della control unit.*
- *Infine collegare i reset di tutte le memorie ed aggiungere il segnale di clock.*

CPU Datapath

Esercizio 04 - Circuito



Estensione

Esercizio x casa

- *Estendere l'architettura realizzata in questa esercitazione ed implementare la seguente istruzione:*
  - *addi X1, X2, imm*

- *Suggerimento: la somma tra immediato e registro è già implementata a livello di datapath, vedere le istruzioni load / store.*

Estensione

Esercizio x casa

➔ *Codificare la nuova istruzione e aggiungerla alla tabella delle istruzioni implementate:*

Formato	Istruzione	Opcode	funct3	funct7	ALU Action	ALU Control
R-Type	add	10	00	00	add	10
	sub	10	00	01	sub	11
	and	10	11	00	and	00
	or	10	10	00	or	01
SB-Type	beq	01	xx	xx	sub	11
I-Type	lb	00	00	xx	add	10
	addi	00	01	xx	add	10
S-Type	sb	11	xx	xx	add	10

Estensione

Esercizio x casa

➔ *Specificare il comportamento della Control Unit rispetto all'istruzione aggiunta:*

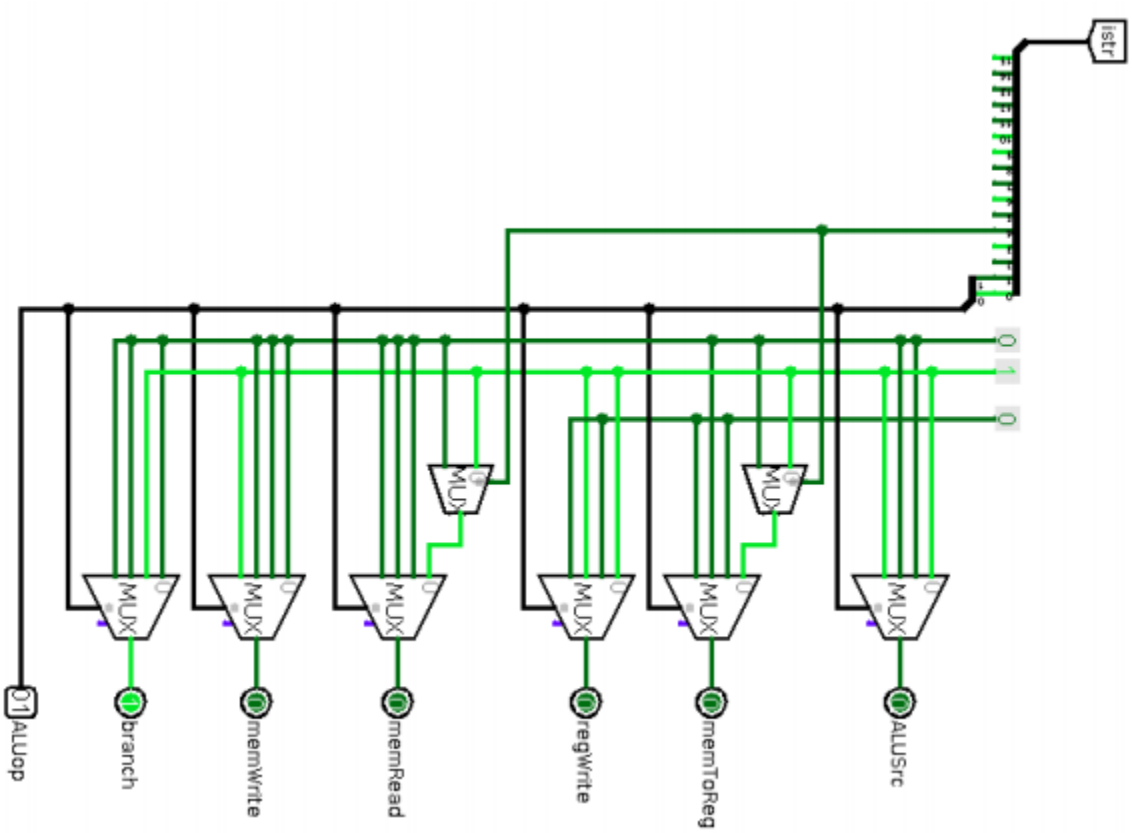
Istruzione	ALUsrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUop1	ALUop0
R-Type	0	0	1	0	0	0	1	0
lb	1	**1**	1	**1**	0	0	0	0
sb	1	x	0	0	1	0	1	1
beq	0	x	0	0	0	1	0	1
addi	1	0	1	0	0	0	0	0

➔ *Notare le differenze rispetto alla load byte.*

Estensione

Esercizio x casa

➔ *Modificare la Control Unit in base a quanto definito in precedenza:*



Programmazione
Esercizio 06

- *Esercizio di riepilogo, attraverso il quale metteremo in pratica tutte le cose viste durante le esercitazioni.*

- *Scrivere un programma assembly che vada ad implementare una somma di vettori, il programma deve essere eseguibile sul processore progettato in questa esercitazione;*

- *Il programma deve essere assemblato a mano e deve essere composto solo dalle istruzioni supportate (add, sub, and, or, lb, sb, addi, beq).*

Programmazione

Esercizio 06 - Soluzione

```
1  .globl main
2
3  .data
4          a: .byte  1, 2, 3, 4
5          b: .byte  5, 6, 7, 8
6          c: .space 4
7  .text
8          main:
9                  addi    s1, zero, 4
10                 addi    s2, zero, 0
11                 la      s3, a
12                 la      s4, b
13                 la      s5, c
14
15                 loop:
16
17                         lb      s6, (s3)
18                         lb      s7, (s4)
19
20                         add     s8, s6, s7
21                         sb      s8, (s5)
22
23                         addi    s3, s3, 1
24                         addi    s4, s4, 1
25                         addi    s5, s5, 1
26
27                         addi    s2, s2, 1
28                         bne     s2, s1, loop
29         li      a7, 93
30         li      a0, 0
31         ecall
```

- *Versione di partenza, che è possibile simulare tramite RARS;*
- *L'output ottenuto è il seguente:*

Address	Value (+0)	Value (+4)	Value (+8)
0x10010000	0x04030201	0x08070605	0x0c0a0806

Programmazione

Esercizio 06 - Soluzione

```
.globl main

.data
        a: .byte  1, 2, 3, 4
        b: .byte  5, 6, 7, 8
        c: .space 4
.text
        main:
                addi    s2, zero, 4
                #addi   s2, zero, 0

                la      s3, a   # usato sia come indice del loop i = 0
                                # sia come puntatore
                #la     s4, b
                #la     s5, c

                loop:   beq s3, s2, endl

                        lb s1, 0(s3)
                        lb s4, 4(s3)

                        #lb s6, (s3)
                        #lb s7, (s4)

                        add s4, s1, s4

                        #add s8, s6, s7
                        #sb s8, (s5)

                        sb s4, 8(s3)

                        #addi s3, s3, 1
                        #addi s4, s4, 1
                        #addi s5, s5, 1

                        addi s3, s3, 1
                        #bne s2, s1, loop
                        beq s1, s1, loop
                endl: beq s1, s1, endl
                #li     a7, 93
                #li     a0, 0
                #ecall
```

Versione con solo le istruzioni supportate dalla nostra architettura;

Al momento non è utilizzabile in RARS in quanto presuppone che «a», «b» e «c» partano dall'indirizzo 0x00000000

Programmazione

Esercizio 06 - Soluzione

```
1   .data
2           a: .byte  1, 2, 3, 4
3           b: .byte  5, 6, 7, 8
4           c: .space 4
5   .text
6           addi    R1, R0, 4
7
8           loop:   beq     R2, R1, endl
9
10                  lb      R0, 0(R2)
11                  lb      R3, 4(R2)
12
13                  add     R3, R0, R3
14
15                  sb      R3, 8(R2)
16
17                  addi    R2, R2, 1
18                  beq     R0, R0, loop
19  endl: beq       R0, R0, endl
```

➔ *In questo caso partiamo dal presupposto che «a» parta dall'indirizzo 0x00000000.*

Programmazione
Esercizio 06 - Soluzione

➔ *Assemblare il segmento dati:*

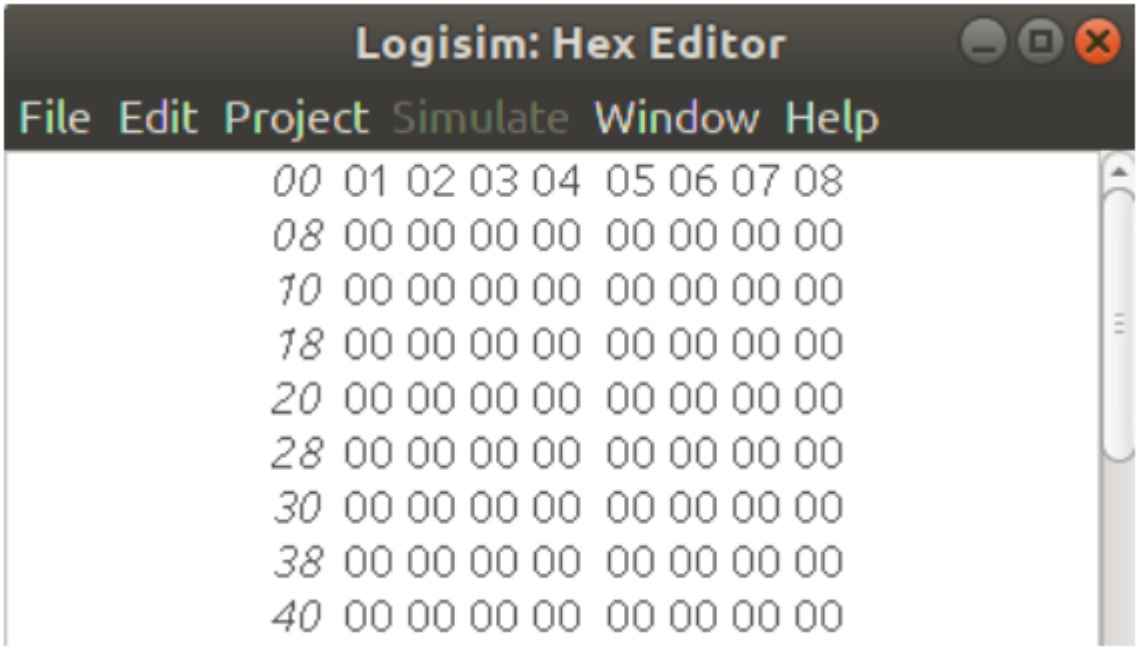
```
1   .data
2           a: .byte  1, 2, 3, 4
3           b: .byte  5, 6, 7, 8
4           c: .space 4
```

➔ *Semplicemente sostituire le label “a”, “b” e “c” con gli indirizzi in memoria:*

```
.data
        0x00: 0x01 0x02 0x03 0x04
        0x04: 0x05 0x06 0x07 0x08
```

Programmazione
Esercizio 06 - Soluzione

➔ *Posizionare infine il contenuto del segmento dati, all'interno della data memory della CPU, come in figura:*



Programmazione
Esercizio 06 - Soluzione

- *Assemblare il segmento text, sostituendo per ogni istruzione il corrispettivo codice esadecimale.*

- *Per esempio:*
- ***addi R1, R0, 4*** *→ Istruzione I-Type, avente il seguente formato:*

15		*…*		*0*
Imm[7:0]	**rs1**	**funct3**	**rd**	**op**
8bit	*2bit*	*2bit*	*2bit*	*2bit*

- *Nel nostro caso diventa:*
- *0000.0100.0001.0100* → ***0x0414***

Programmazione

Esercizio 06 - Soluzione

- ➔ *Procedere in questo modo per tutte le altre istruzioni del programma.*
- ➔ *Tenere a mente che per le istruzioni di salto (beq), la label va tradotta in un indirizzo relativo, ad esempio:*

- ➔ ***beq R2, R1, endl*** ➔ *in questo caso la label «endl» è posizionata 7 istruzioni dopo, quindi in caso di salto, il program counter si deve incrementare di 7.*

- ➔ *1000.0110.0100.1001* ➔ ***0x8649***

15		…			*0*
Imm[0, 7, 5:2]	**rs2**	**rs1**	**funct3**	**Imm[1, 6]**	**op**
6bit	*2bit*	*2bit*	*2bit*	*2bit*	*2bit*

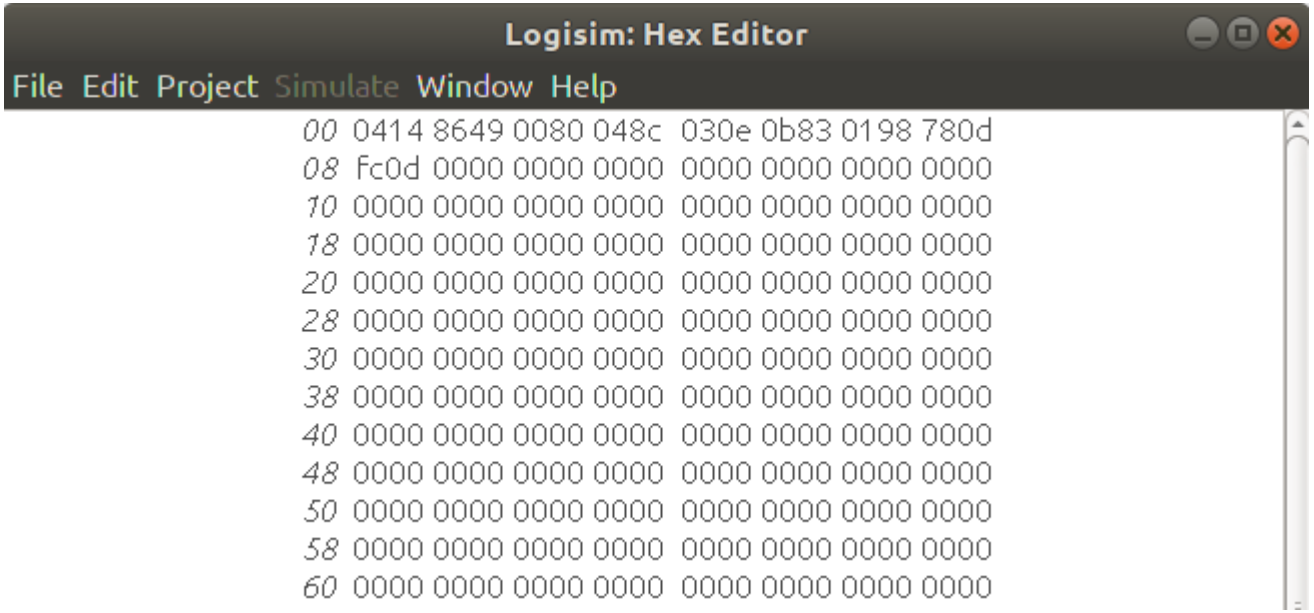
Programmazione

Esercizio 06 - Soluzione

```
1   .data
2           0x00:  0x01 0x02 0x03 0x04
3           0x04:  0x05 0x06 0x07 0x08
4
5   .text
6           0x00:    0x0414
7           0x01:    0x8649
8           0x02:    0x0080
9           0x03:    0x048C
10          0x04:    0x030E
11          0x05:    0x0B83
12          0x06:    0x0198
13          0x07:    0x780D
14          0x08:    0xFC0D
```

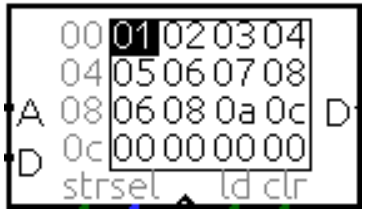
Programmazione
Esercizio 06 - Soluzione

➔ *Inizializzare infine anche le istruzioni assemblate all'interno della instruction memory:*



Programmazione
Esercizio 06 - Soluzione

➔ *Memoria dati dopo l'esecuzione del programma:*



➔ *Notiamo i due vettori che abbiamo inizializzato (0x00, 0x04).*

➔ *Ed il vettore somma posizionato all'indirizzo 0x08.*