Instruction Set Architecture

Esercitazione
Architettura degli elaboratori

Esercizio 1

Data l’istruzione

j xx

occupa più spazio in memoria la sua rappresentazione secondo codifica ASCII o l’istruzione macchina corrispondente dopo che è stata assemblata?

Esercizio 1 – soluzione

- La rappresentazione **ASCII** dell'istruzione richiede 32 bit: è composta da 4 caratteri, ciascuno rappresentato con un byte
- Dopo che è stata assemblata, l'istruzione macchina richiede (come ogni altra istruzione MIPS), 32 bit

In questo caso particolare, quindi, la codifica ASCII e la corrispondente istruzione macchina occupano lo stesso numero di bit

Esercizio 2

Determinare a quale istruzione macchina MIPS corrisponde la sequenza binaria

00100010111010110000000001100000

Esercizio 2 – procedimento

1. cerco l'opcode (conversione in decimale o esadecimale dei primi 6 bit) nella tabella a pagina 50 dell'appendice A - in alcuni casi saranno necessari anche gli ultimi 6 bit (function code)

ATTENZIONE! Nella tabella a pag. A-50, op[31:26] è indicato in decimale o esadecimale mentre func[0:5] è SOLO in decimale!

2. cerco nell'appendice A la descrizione (sintassi e semantica) dell'istruzione corrispondente
   - dalla descrizione della sintassi capisco il formato (e tipo) dell'istruzione che servirà per sapere come dividere i restanti bit
   - dalla descrizione della semantica dell'istruzione capisco come interpretare i restanti bit

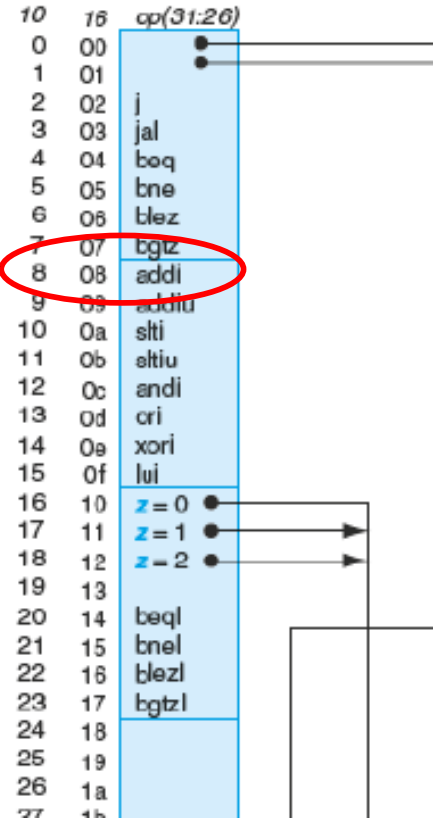
Esercizio 2 – soluzione

001000 10111010110000000001100000

Opcode= $001000_2 = 08_{16} = 8_{10}$

1

pag. 50, App.A



Esercizio 2 – soluzione

001000 10111010110000000001100000

Opcode= $001000_2 = 08_{16} = 8_{10}$

1

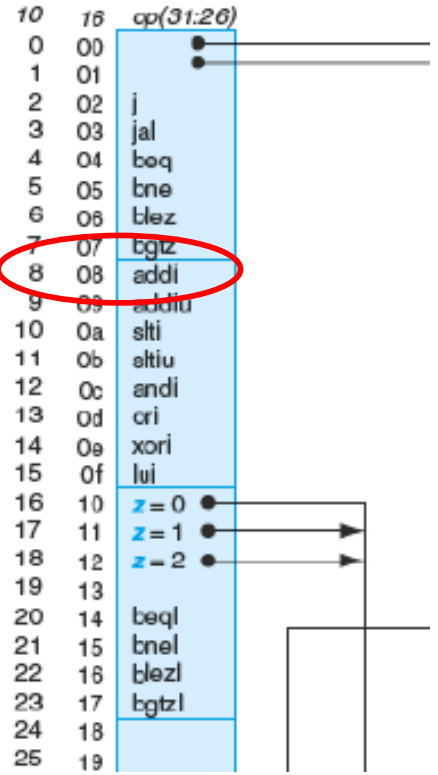
addi → l'istruzione è nel formato I-type

001000 10111 01011 0000000001100000

addi rs (5 bit) rt (5 bit) immediate (16 bit)

il tipo lo vedo dalla descrizione della sintassi dell'istruzione

pag. 50, App.A



addi, in App.A (pag. A-51)

Addition immediate (with overflow)

```
addi rt, rs, imm
```

8	rs	rt	imm
6	5	5	16

2 Put the sum of register `rs` and the sign-extended immediate into register `rt`.

Esercizio 2 – soluzione

001000 10111010110000000001100000

Opcode= $001000_2 = 08_{16} = 8_{10}$

1

addi → l'istruzione è nel formato I-type

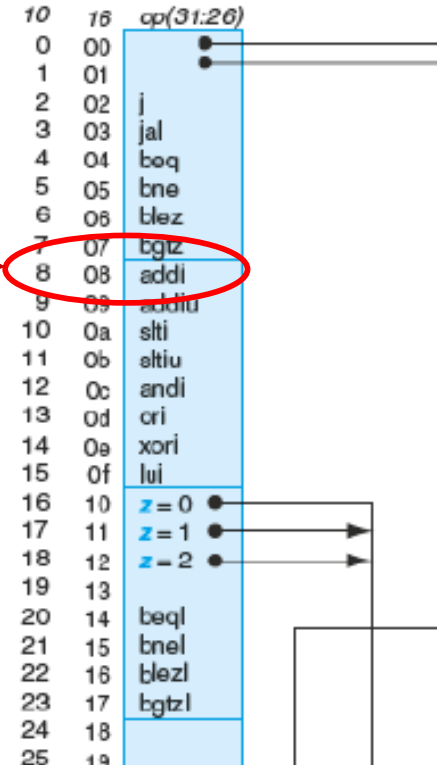
il tipo lo vedo dalla descrizione della sintassi dell'istruzione

001000 10111 01011 0000000001100000

addi rs (5 bit) rt (5 bit) immediate (16 bit)

addi $11, $23, 96

pag. 50, App.A



addi, in App.A

Addition immediate (with overflow)

addi rt, rs, imm

8	rs	rt	imm
6	5	5	16

2 Put the sum of register rs and the sign-extended immediate into register rt.

Esercizio 3

A quale istruzione macchina MIPS corrisponde il codice esadecimale:

0x8fa40000

Esercizio 3 – soluzione

Esadecimale: 0x8fa40000

Binario: 1000 1111 1010 0100 0000 0000 0000 0000

opcode= 35_{10}= 23_{16}

1) da opcode[31:26]
a pag.A-50 App A → lw

2) sintassi e semantica di lw a pag. A-67

Load word

`lw rt, address`

0x23	rs	rt	Offset
6	5	5	16

Load the 32-bit quantity (word) at *address* into register `rt`.

modalità di indirizzamento della memoria:
c(rx), che utilizza come indirizzo la somma
della costante c e del contenuto del registro rx

Quindi, l'istruzione

100011 11101 00100 0000000000000000

corrisponde a lw $4, 0($29)

Esercizio 4

A quale istruzione macchina MIPS corrisponde il codice esadecimale:

0x0232502A

Esercizio 4 – soluzione

Esadecimale: 0x0232502A

Binario: 0000 0010 0011 0010 0101 0000 0010 1010

opcode= 0 funct=42

pag.A-57 App A → slt

Set less than

`slt rd, rs, rt`

0	rs	rt	rd	0	0x2a
6	5	5	5	5	6

Set register `rd` to 1 if register `rs` is less than `rt`, and to 0 otherwise.

Formato R-type:

000000 10001 10010 01010 00000 101010

slt $10, $17, $18

Esercizio 5

- Scrivere l'istruzione MIPS che effettua la differenza tra i valori contenuti nei registri $3 e $4, e deposita il risultato nel registro $2

- Qual è il formato dell'istruzione?

- Qual è la rappresentazione binaria ed esadecimale dell'istruzione?

Esercizio 5 – soluzione

- L'istruzione MIPS che effettua la differenza tra i contenuti di due registri è

Subtract (with overflow)

`sub rd, rs, rt`

0	rs	rt	rd	0	0x22
6	5	5	5	5	6

Put the difference of registers `rs` and `rt` into register `rd`.

Appendice A, pag.A-56

- In questo caso,
  - $rs= $3
  - $rt = $4
  - $rd = $2
- Quindi l'istruzione è **sub $2, $3, $4**
- Il formato dell'istruzione (ricavabile dalla descrizione della sintassi a pag. A-56)

  R-type → [op:6][rs:5][rt:5][rd:5][shamt:5][funct:6]
- La rappresentazione binaria dell'istruzione è

  000000 00011 00100 00010 00000 100010 **(opcode=0, funct=0x22=34_{10})**
- La rappresentazione esadecimale dell'istruzione

  0000 0000 0110 0100 0001 0000 0010 0010

  è quindi 0 0 6 4 1 0 2 2

Esercizio 6

- Scrivere l'istruzione MIPS che effettua la somma del valore memorizzato nel registro $6 e del valore costante 100, e deposita il risultato nel registro $5

- Qual è il formato dell'istruzione?

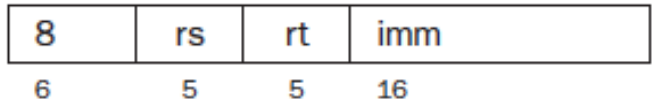
- Qual è la rappresentazione binaria ed esadecimale dell'istruzione?

Esercizio 6 – soluzione

- L'istruzione MIPS che effettua la somma tra il contenuto di un registro e un valore costante e deposita il risultato in un registro è

Addition immediate (with overflow)

`addi rt, rs, imm`



(opcode=08)

Put the sum of register `rs` and the sign-extended immediate into register `rt`.

- In questo caso,
  - $rs= $6
  - $rt = $5
  - imm = 100

- Quindi l'istruzione è **addi $5, $6, 100**
- Il formato dell'istruzione è **I-type → [op:6][rs:5][rt:5][constant:16]**
- La rappresentazione binaria dell'istruzione è **0010 0000 1100 0101 0000 0000 0110 0100**
- La rappresentazione esadecimale è **0x20C50064**

Esercizio 7

Considerando che i campi di una istruzione R-type sono i seguenti:

- **opcode**: 0x0
- **rs**: 0x18
- **rt**: 0x19
- **rd**: 0x1B
- **shamt**: 0x0
- **funct**: 0x24

indicare l'istruzione e la sua corrispondente rappresentazione in esadecimale e in binario

- Dovremo prima cercare l'istruzione corrispondente a opcode e function code e quindi, vista la sintassi, sostituire le varie parti

Esercizio 7 – soluzione

- I campi opcode e funct sono 0x0 e 0x24 (36 in decimale) → ricavo dalle colonne op[31:26] e func[0:5] della tabella a pag. A-50 che l'istruzione è una **and** - ATTENZIONE! op[31:26] è in decimale o esadecimale mentre func[0:5] è SOLO in decimale!
- La sintassi dell'istruzione **and** (pag. A-52) è **and $rd, $rs, $rt**
- Essendo in questo caso
  - **rs**: 0x18 → $24_{10} = 11000_2$
  - **rt**: 0x19 → $25_{10} = 11001_2$
  - **rd**: 0x1B → $27_{10} = 11011_2$

abbiamo che l'istruzione corrispondente è **and** $27, $24, $25

AND

`and rd, rs, rt`

0	rs	rt	rd	0	0x24
6	5	5	5	5	6

Put the logical AND of registers `rs` and `rt` into register `rd`.

- La rappresentazione in binario dell'istruzione è

  000000 11000 11001 11011 00000 100100

- La rappresentazione in esadecimale dell'istruzione

  0000 0011 0001 1001 1101 1000 0010 0100 (ricavo la rappresentazione esadecimale convertendo in cifre esadecimali gruppi di quattro bit)

  è 0 3 1 9 D 8 2 4

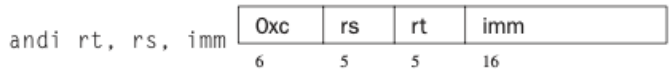
Esercizio 8

- Considerando che i campi di una istruzione I-type sono i seguenti:
  - **opcode**: 0xC
  - **rs**: 0x10
  - **rt**: 0xF
  - **imm:** 0x1
- Indicare l'istruzione e la sua corrispondente rappresentazione in esadecimale e in binario
- .

Esercizio 8 – soluzione

- Il campo opcode è 0xC → ricavo dalla prima colonna della tabella a pag. A-50 che l'istruzione è una **andi**
- La sintassi dell'istruzione **andi** (pag. A-52) è **andi $rt, $rs, imm**
- Essendo in questo caso
  - **rs**: 0x10 → 16_{10} = 10000
  - **rt**: 0xF → 15_{10} = 01111
  - **imm:** 0x1 → 1_{10} = 0000000000000001

AND immediate



Put the logical AND of register `rs` and the zero-extended immediate into register `rt`.

- L'istruzione è **andi $15, $16, 1**
- La sua rappresentazione in binario è 00110010000011110000000000000001
- In esadecimale 3 2 0 F 0 0 0 1

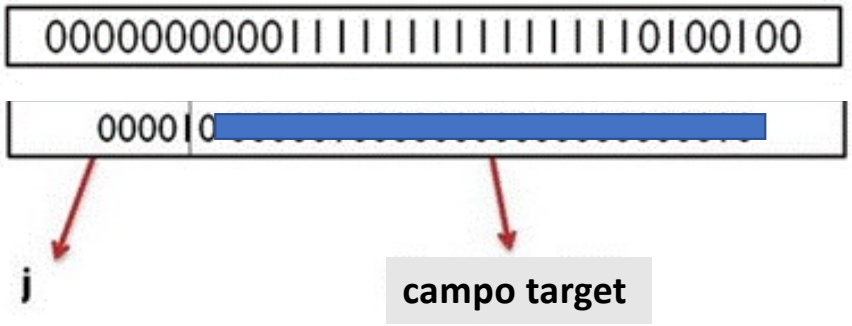
ottenuta dalla codifica in cifre esadecimali dei gruppi di 4 bit

0011 0010 0000 1111 0000 0000 0000 0001

Esercizio 9 – jump (salto incondizionato)

Se Program Counter (PC) contiene

Instruction Register (IR) contiene

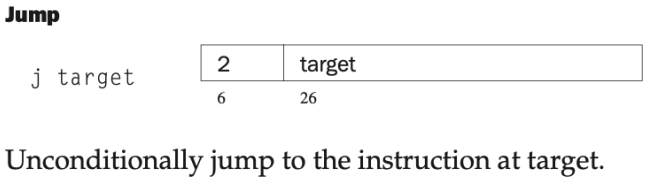


e il programma in memoria è

Indirizzo	Istruzione
0x00400000	add $9,....
0x00400004	sub $14, ...
0x00400008	lw $8,...
0x0040000C	add $7,...

qual è il valore del campo target dell'istruzione di salto incondizionato in esecuzione affinchè la prossima istruzione eseguita sia l'istruzione lw $8, …?

Esercizio 9 – soluzione



- Istruzione “j target” cambia il valore del PC e l’esecuzione del programma “salta/passa” all’indirizzo di memoria (32 bit) che si ottiene dal campo target di 26 bit:
  - aggiungendo i primi 4 bit del contenuto del PC a sinistra (parte più significativa)
  - aggiungendo 2 bit a 0 come parte meno significativa (tutte le istruzioni si trovano ad indirizzi multipli di 4)
- In questo caso, il nuovo valore del PC perché venga eseguita l‘istruzione che si trova all’indirizzo 0x00400008 (= 0000 0000 0100 0000 0000 0000 0000 1000) dovrà essere 0x00400008
- Quindi l’istruzione jump nel registro IR dovrà avere il campo target uguale all’indirizzo a cui si vuole saltare togliendo gli ultimi 2 bit e i primi 4 bit (*se questi ultimi coincidono con i primi 4 del PC, altrimenti il salto richiesto non sarà possibile*)
- Il campo target dell’istruzione jump nel IR (con il contenuto di PC indicato dall’esercizio 0000xxxxxxxxxxxxxxxxxxxxxxxxxxxx) dovrà essere

  0000 0100 0000 0000 0000 0000 10

Esercizio 10

Qual è il valore esadecimale dell'indirizzo della più lontana cella di memoria a cui è possibile saltare con una istruzione di salto incondizionato (**jump**) se il contenuto del registro PC è 00110010010010110010000100011000?

Esercizio 10 – soluzione

- La sintassi dell'istruzione di salto incondizionato è `j target`
- L'indirizzo (di 32 bit) dell'istruzione verso cui avviene il salto è dato da
  - i primi **4 bit del** Program Counter
  - **concatenati** ai **26 bit di target**
  - **concatenati** a **00** (ogni istruzione si trova ad un indirizzo multiplo di 4)
- In questo caso: PC = 00110010010010011001000010001100 0
  - primi 4 bit di PC = 0011
  - valore massimo che ci permette di specificare l'istruzione `j target` ha i **26 bit di target a 1** (con tutti i 26 bit a 0 avremmo un salto ad una distanza inferiore)
- L'indirizzo della più lontana cella di memoria a cui è possibile saltare con un istruzione di salto incondizionato da PC=00110010010010011001000010001100 0 è

  0011 1111 1111 1111 1111 1111 1111 1100
- In esadecimale: 0x3FFFFFFC

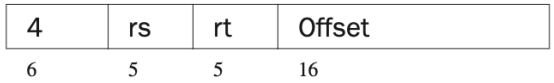
Esercizio 11 – branch (salti condizionati)

Scrivere in linguaggio macchina l'istruzione assembly MIPS che salta 12 istruzioni se i contenuti dei registri $12 e $15 sono uguali

Esercizio 11 – soluzione

Branch on equal

`beq rs, rt, label`



Conditionally branch the number of instructions specified by the offset if register `rs` equals `rt`.

- Supponendo che 12 istruzioni dopo quella corrente si trovi il simbolo/etichetta dodicidopo, l'istruzione assembly MIPS che salta 12 istruzioni se i contenuti dei registri $12 e $15 sono uguali è

  beq $12, $15, dodicidopo

- In linguaggio macchina l'istruzione è

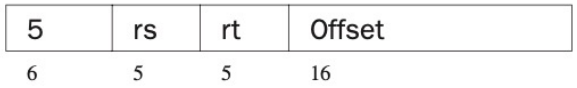
  00010001100011110000000000001100

Esercizio 12

Qual è l'indirizzo della più lontana cella di memoria a cui è possibile saltare (in avanti) con una istruzione "bne" se il PC contiene 00100001011000011011000100100000 ?

Branch on not equal

`bne rs, rt, label`



Conditionally branch the number of instructions specified by the offset if register `rs` is not equal to `rt`.

Esercizio 12 – soluzione

- I salti condizionati sono salti relativi alla posizione attuale (indicata da PC)
- L'indirizzo della più lontana cella di memoria a cui possiamo *saltare* ***in avanti*** con un'istruzione di salto condizionato è ottenuta sommando al contenuto del PC, il *numero* ***più grande positivo rappresentabile in CA2*** con 16 bit (offset o branch address dell'istruzione di branch) concatenati con due bit a 0
- In questo caso quindi

(PC) 00100001011000011011000100100000 +

(max branch address positivo * 4) 00000000000000011111111111111100 =

00100001011000101011000100011100

Esercizio 13 – accesso in memoria (load e store)

- Indicare le istruzioni per calcolare la somma del valore che è memorizzato nel registro $17, e del valore memorizzato nella locazione di memoria che si trova 14 parole di memoria più avanti rispetto all'indirizzo specificato dal contenuto del registro $16
- Memorizzare il risultato 3 parole di memoria più avanti rispetto alla locazione attuale del secondo operando

Esercizio 13

Dovremo prima caricare in un registro il valore da sommare e una volta effettuato il calcolo e depositato il risultato in un registro, salvare in memoria il contenuto di tale registro

Useremo

- una istruzione **lw $rs, offset($rt)** per scrivere in $rs il contenuto della locazione di memoria che si trova dopo l'indirizzo base (indicato dal contenuto del registro $rt) di un numero di byte indicato da offset
- una istruzione **sw $rs, offset($rt)** per scrivere il contenuto del registro $rs in memoria e all'indirizzo che si trova dopo l'indirizzo base (indicato dal contenuto di $rt) di un numero di byte pari a offset

Esercizio 13 – soluzione

In questo caso vogliamo:

- sommare il contenuto del registro $17 ad un valore memorizzato 14 word (quindi 56=14*4 byte) dopo l'indirizzo indicato dal contenuto del registro $16
- salvare il risultato della somma ad un indirizzo 3 word (12 byte) dopo il valore letto (e sommato)

Quindi:

- **lw $8, 56($16)** **#scrivo in un registro temporaneo (e.g. $8) il valore in memoria che si trova 56 byte (14 word) dopo l'indirizzo indicato dal contenuto di $16**
- **add $9, $17, $8** **#sommo il contenuto del registro $8 con il contenuto di $17 e lo deposito temporaneamente in un registro (e.g. $9)**
- **sw $9, 68($16)** **#memorizzo il risultato nella somma (ora in $9) in memoria all'indirizzo che si trova 12 byte (=3 word) dopo il valore letto**

Esercizio 14 – istruzioni native

Indicare **una** istruzione nativa MIPS per:

- azzerare il contenuto del registro $2

Indicare **una** istruzione nativa MIPS per:

- scrivere il contenuto del registro $2 nel registro $1?

Esercizio 14 – istruzioni native

Indicare **una** istruzione nativa MIPS per:

- azzerare il contenuto del registro $2

Indicare **una** istruzione nativa MIPS per:

- scrivere il contenuto del registro $2 nel registro $1

Non è possibile utilizzare

Load immediate

`li rdest, imm` *pseudoinstruction*

Move the immediate `imm` into register `rdest`.

Move

`move rdest, rsrc` *pseudoinstruction*

Move register `rsrc` to `rdest`.

Esercizio 14 – soluzione

- per azzerare il contenuto del registro $2, è possibile usare:
  - **add $2, $zero, $zero**

  scrive in $2 il risultato della somma della costante 0 (contenuta nel registro $zero) con se stessa

  oppure
  - **and $2, $2, $zero**

  scrive in $2 il risultato dell'operazione logica AND tra l'attuale contenuto di $2 e la costante 0 (contenuta nel registro $zero)

- per scrivere il contenuto del registro $2 nel registro $1, è possibile utilizzare

  add $1, $2, $zero

  che scrive in $1 il risultato della somma del contenuto del registro $zero (costante 0) e il contenuto del registro che vogliamo copiare $2

Esercizio 15

Supponendo che un valore ***a*** si trovi nel registro $8

- Indicare **due** istruzioni native MIPS che scrivano nel registro $9 il risultato di 3*a (a+a+a)

- Indicare **due** istruzioni native MIPS che scrivano nel registro $9 il risultato di 4*a (a+a+a+a)

Esercizio 15 – soluzione

Se il valore a si trova nel registro $8

- Per scrivere in $9 il risultato di a*3 (a+a+a), è possibile utilizzare le due operazioni di somma:

```
add $9, $8, $8        #in $9 avrò a+a
add $9, $9, $8        #in $9 avrò 2a+a
```

- Per scrivere in $9 il risultato di a*4 (a+a+a+a) , è possibile utilizzare le due operazioni di somma:

```
add $9, $8, $8        #in $9 avrò a+a=2a
add $9, $9, $9        #in $9 avrò 2a+2a
```

Esercizio 16 – cicli

Quattro valori, della dimensione di una word ciascuna, sono memorizzati di seguito in memoria a partire dall'indirizzo specificato dal contenuto del registro $10.

- Scrivere la sequenza di istruzioni assembly che aggiunge la costante 10 ai quattro valori e poi li salva al medesimo indirizzo di memoria cui si trovavano.

N.B. Per la lettura dei valori successivo al primo, sarà necessario incrementare l'offset di 4 byte (per ciascuna word) **oppure** incrementare il contento del registro che contiene l'indirizzo base.

La prima di queste opzioni NON permette di realizzare un ciclo mentre la seconda sì

Esercizio 16 – soluzione a

```
# soluzione con incremento dell'offset di 4 unità per ciascuna delle word da leggere
lw $8, 0($10)          #scrivo in $8 il primo valore – N.B. offset è 0
addi $8, $8, 10
sw $8, 0($10)
lw $8, 4($10)          #scrivo in $8 il secondo valore – N.B. offset è 4
addi $8, $8, 10
sw $8, 4($10)
lw $8, 8($10)          #scrivo in $8 il terzo valore  – N.B. offset è 8
addi $8, $8, 10
sw $8, 8($10)
lw $8, 12($10)         #scrivo in $8 il quarto valore – N.B. offset è 12
addi $8, $8, 10
sw $8, 12($10)
```

Esercizio 16 – soluzione b

#medesimo comportamento ma questa soluzione si presta alla scrittura di un ciclo – con la versione precedente NON è possibile!

```
lw $8, 0($10)          #scrivo in $8 il primo valore – N.B. offset è 0
addi $8, $8, 10
sw $8, 0($10)          #scrivo in memoria $8, allo stesso indirizzo (scritto in $10 – N.B. offset è 0)
addi $10, $10, 4       #aggiungo 4 all'indirizzo contenuto in $10 (il secoondo valore si trova 1 word=4 byte dopo)
lw $8, 0($10)          #scrivo in $8 il secondo valore – N.B. offset è 0
addi $8, $8, 10
sw $8, 0($10)          #scrivo in memoria $8, allo stesso indirizzo (scritto in $10 – N.B. offset è 0)
addi $10, $10, 4       #aggiungo 4 all'indirizzo contenuto in $10 (il terzo valore si trova 1 word=4 byte dopo)
lw $8, 0($10)          #scrivo in $8 il terzo valore – N.B. offset è 0
addi $8, $8, 10
sw $8, 0($10)          #scrivo in memoria $8 , allo stesso indirizzo (scritto in $10 – N.B. offset è 0)
addi $10, $10, 4       #aggiungo 4 all'indirizzo contenuto in $10 (il quarto valore si trova 1 word=4 byte dopo)
lw $8, 0($10)          #scrivo in $8 il quarto valore letto all'indirizzo scritto in $10 – N.B. offset è 0
addi $8, $8, 10
sw $8, 0($10)
```

Esercizio 16 – soluzione b (con ciclo)

#medesimo comportamento ma utilizzando un ciclo

```
addi $9, $zero, 4          #inizializzo a 4 il contenuto del registro $9 (numero di cilci da eseguire)
add $11, $zero, $zero      #inizializzo a 0 il contenuto del registro $11
                           #(contatore del numero di valori da leggere)
ciclo:  lw $8, 0($10)      #scrivo in $8 il primo valore – N.B. offset è 0
        addi $8, $8, 10
        sw $8, 0($10)      #scrivo in memoria $8, allo stesso indirizzo (scritto in $10 – N.B. offset è 0)
        addi $10, $10, 4       #incremento di 4 byte l'indirizzo contenuto in $10
                               #(il valore successive si trova 1 word=4 byte dopo)
        addi $11, $11, 1       #incremento $11 di 1
        bne $11, $9, ciclo     #ripeto il ciclo se non ho raggiunto il numero di valori da leggere
```