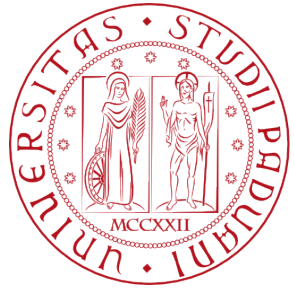
CPUSim
Laboratorio 8/1/2016.

Francesco Tapparo

e-mail: tapparo@math.unipd.it



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Wombat 2.1 - Instruction set

INSTRUCTION	MEANING	DESCRIPTION
readR reg1	Read n $\rightarrow$ reg1	Input from keyboard in reg1
writeR reg1	reg1 $\rightarrow$ output	Write value of reg1
multiplyR reg1 reg2 reg3	reg1 * reg2 $\rightarrow$ reg3	Multiply contents of two registers
divideR reg1 reg2 reg3	reg1 / reg2 $\rightarrow$ reg3	Divide contents of two registers
subtractR reg1 reg2 reg3	reg1 - reg2 $\rightarrow$ reg3	Subtract contents of two registers
addR reg1 reg2 reg3	reg1 + reg2 $\rightarrow$ reg3	Add contents of two registers
loadR reg1 addr	Mem[addr] $\rightarrow$ reg1	Load word from memory in reg1
storeR reg1 addr	reg1 $\rightarrow$ Mem[addr]	Store word in memory from reg1
jmpzR reg1 addr	If $reg1 = 0$ jump to addr	Conditional jump (reg1=0)
jmpnR reg1 addr	If $reg1 < 0$ jump to addr	Conditional jump ($reg1 < 0$)
jump addr	jump to addr	
stop	stop execution	

Esercizio 2 - Wombat 2.1

Scrivere un programma ASSEMBLY per la CPU Wombat2.1 che legge un intero in ingresso e ritorna 0 se l'intero è pari, -1 altrimenti (usare divisione intera e moltiplicazione).

Esercizio 2 - Soluzione

Listing 1: Esercizio 2

```
readR 1             ; n -> Reg[1]
loadR 2 due         ; 2 -> Reg[2]
divideR 1 2 3       ; Reg[1]/Reg[2] -> Reg[3]
multiplyR 3 2 3     ; Reg[3]*Reg[2] -> Reg[3]
subtractR 3 1 3     ; Reg[3] - Reg[1] -> Reg[3]
writeR 3            ; visualizza Reg[3]
stop                ; termina il programma
due: .data 2 2      ; variabile due
```

Esercizio 3

Scrivere un programma ASSEMBLY per la CPU Wombat2.1 che calcola e stampa il valore assoluto di un intero ricevuto in input. Ancora molto simile a Wombat1.

Esercizio 3 - Soluzione

Listing 2: Esercizio 3

```
readR  1                ;input -> Reg[1]
jmpnR 1 negativo        ;se Reg[1] <0 vai a negativo
fine: writeR 1          ;Reg[1] -> output
stop                    ;stop
negativo:loadR 0 zero   ;Reg[0] = 0
subtractR 0 1 1         ;Reg[0] - Reg[1] -> Reg[1]
jump fine               ;va alla fine
zero: .data 2 0         ;2 byte dove mettere zero
```

Esercizio 4

Scrivere un programma ASSEMBLY per la CPU Wombat2.1 che calcola il prodotto di due interi usando somme.

Esercizio 4 - Soluzione

Listing 3: Esercizio 4

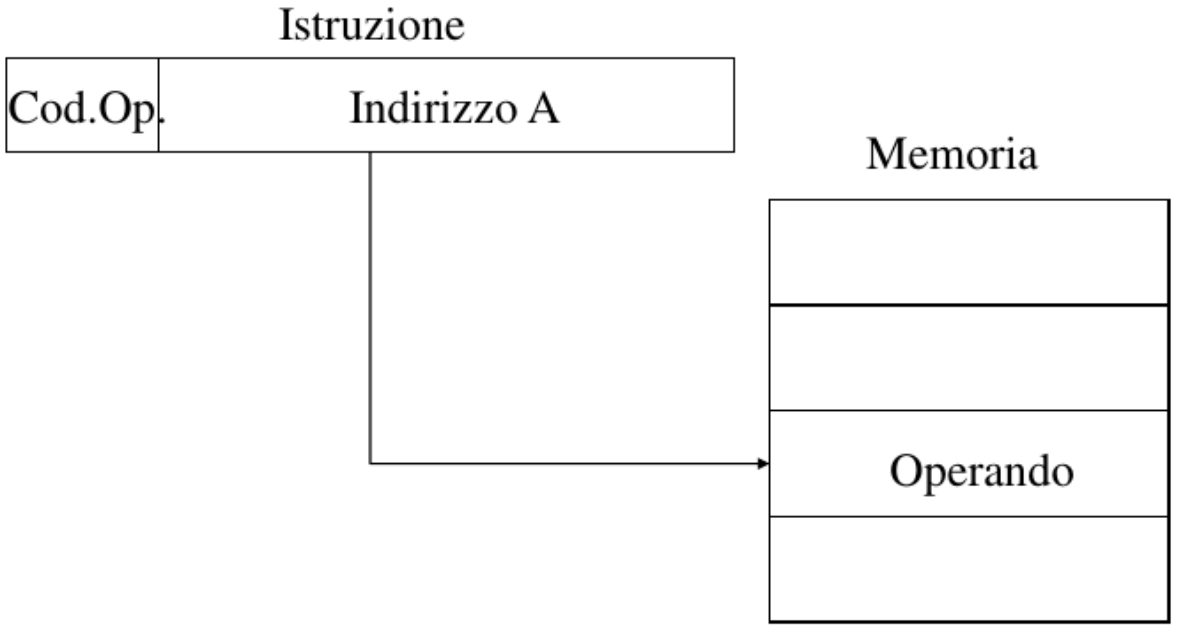
```
readR 0                 ; moltiplicando -> Reg[0]
readR 2                 ; moltiplicatore -> Reg[2]
loadR 1 uno             ; uno -> Reg[1]
ciclo: jmpzR 2 fine     ; vai a fine se Reg[2]=0
addR 0 3 3              ; Reg[0] + Reg[3] -> Reg[3]
subtractR 2 1 2         ; Reg[2] - Reg[1] -> Reg[2]
jump ciclo              ; vai a ciclo
fine: writeR 3          ; output Reg[2]
stop
uno: .data 2 1;
```

Modi di indirizzamento

- Immediato (operando è parte dell'istruzione)
- Diretto (campo indirizzo = indirizzo dell'operando)
- Indiretto (campo indirizzo= indirizzo di una cella di Mem, che contiene l'indirizzo dell'operando)
- Registro (l'operando è in un registro specificato nel campo indirizzo)
- Registro indiretto (campo indirizzo= nome di un registro, che contiene l'indirizzo in Mem dell'operando)
- Spiazzamento (campo indirizzo= A: valore di base (diretto) + R: nome di un registro che contiene un valore da sommare ad A per ottenere l'indirizzo)
- Pila (sequenza lineare di locazioni riservate di M, registro Stack pointer che contiene l'indirizzo alla cima della pila, operando sempre in cima)

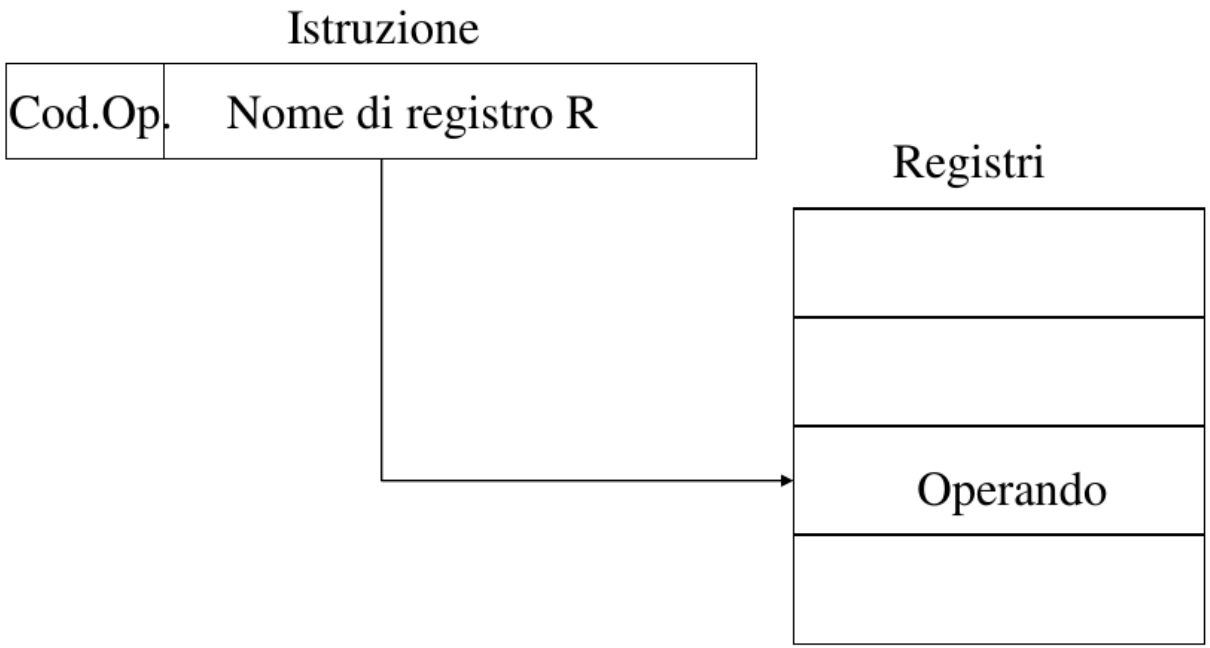
Modi di indirizzamento visti in lab - 1

- Diretto (campo indirizzo = indirizzo dell'operando)
  - CPU Wombat1
  - e.g. ADD X (acc + Mem[X] = acc)
  - 1 accesso alla memoria all'indirizzo X



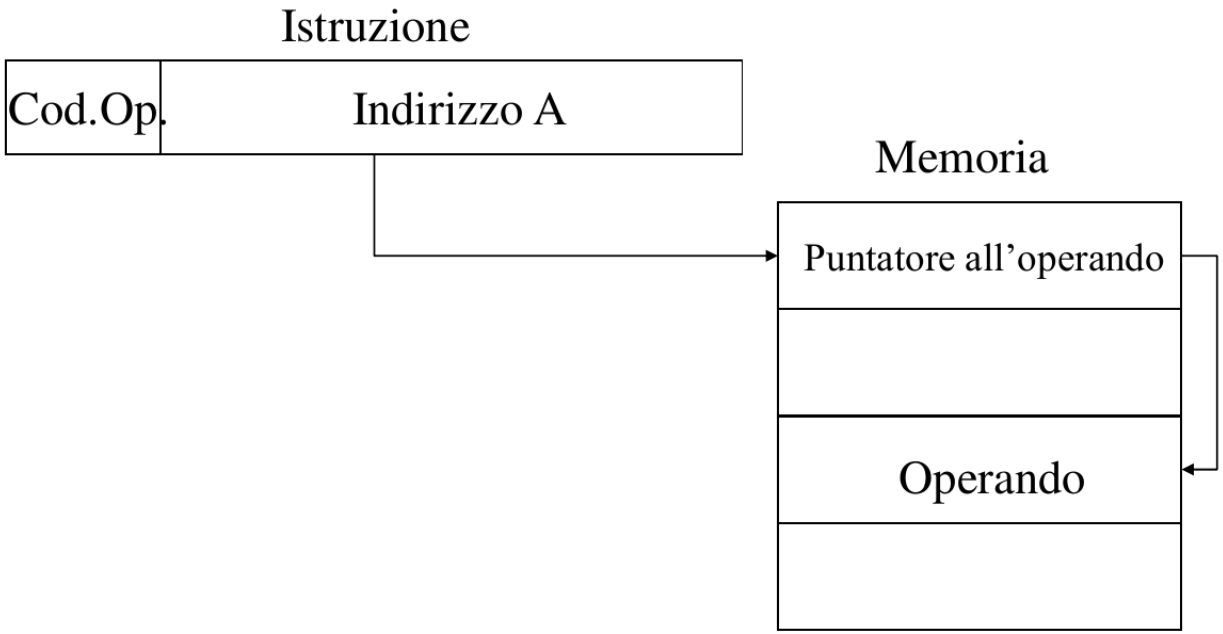
Modi di indirizzamento visti in lab - 2

- Registro (l'operando è in un registro specificato nel campo indirizzo)
  - CPU Wombat2

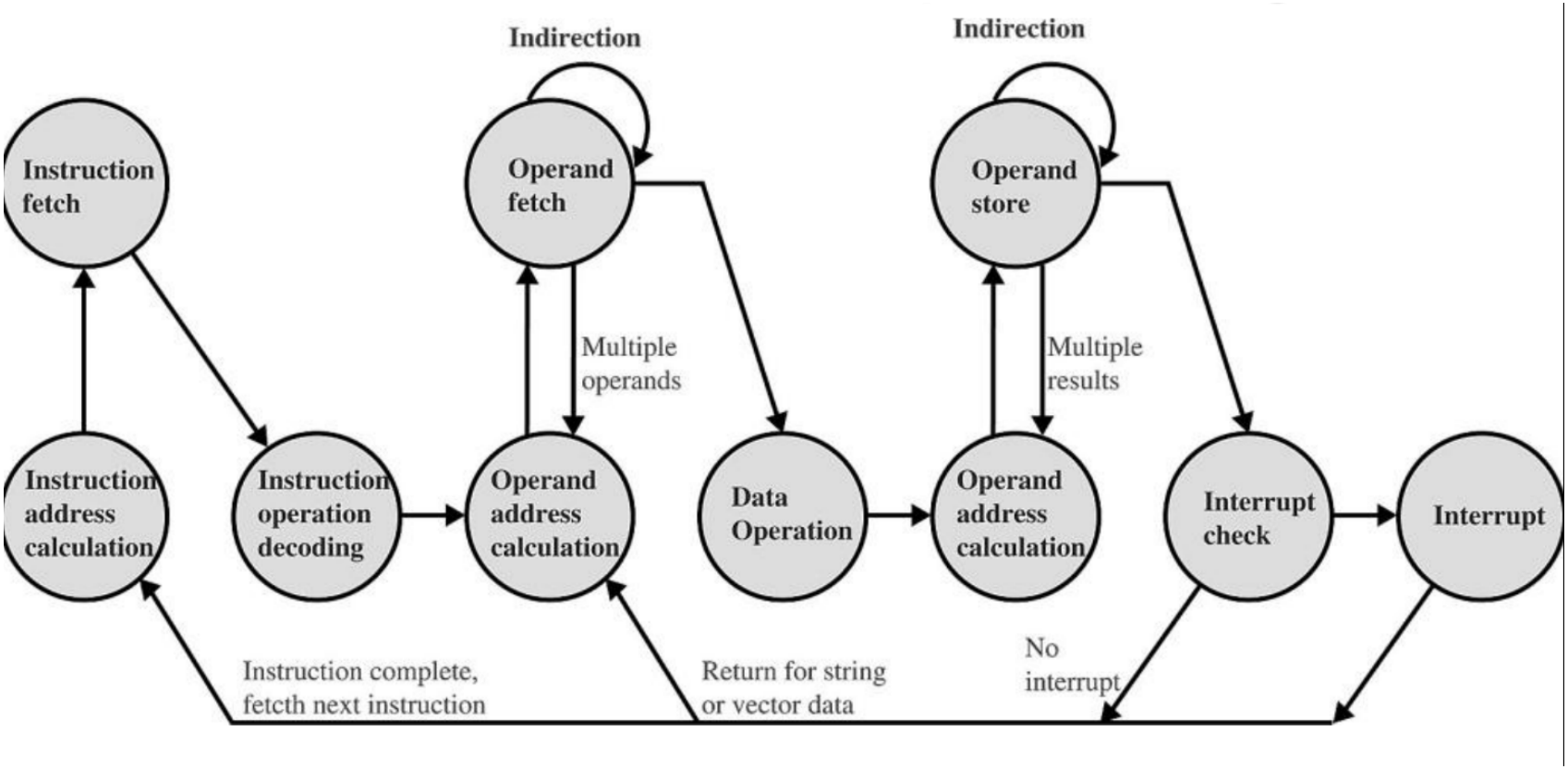


Oggi vedremo...

- Indiretto (campo indirizzo= indirizzo di una cella di Mem, che contiene l'indirizzo dell'operando)
  - e.g.

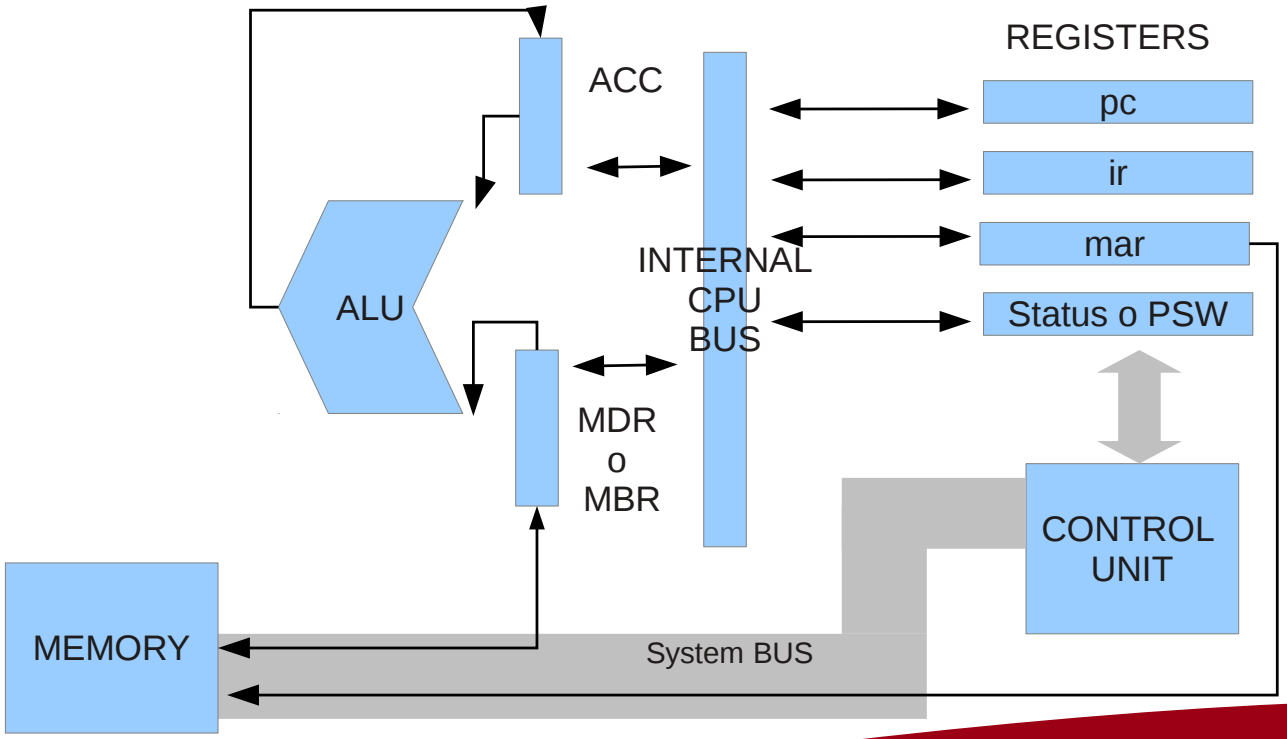


Ind. indiretto - Ciclo di esecuzione



Indirizzamento indiretto: Wombat3

- Apriamo Wombat1.cpu
- Salviamo con un nuovo nome, e.g. Wombat3.cpu
- Se abbiamo altre istruzioni oltre quelle di default, le cancelliamo (opcode da B in su)
- Recall architettura Wombat1:



Indirizzamento indiretto: Wombat3

- Definire nuove istruzioni che interpretino gli operandi:
  - *non* come indirizzi di memoria da caricare, ma
  - come indirizzi degli indirizzi di memoria da caricare.
  - sintassi: **op**(4 bit)+ **address** (del puntatore,12 bit)
  - Servirà una nuova microistruzione:
    - $mdr(4-15)->mar$;
    - copia i 12 bit meno significativi di mdr in mar.
  - Definiamo *Load, Store e Add* con indirizzamento indiretto.

Nuove Istruzioni

- **loadInd**
  - *load indiretta*: addr è l'indirizzo dell'indirizzo della locazione di memoria dove caricare il contenuto di *accumulatore*.
  - *Modify* → *MachineInstructions*
  - Duplichiamo la *load* e la modifichiamo:

Listing 4: loadInd

```
ir(4-15)->mar
Main[mar]->mdr
mdr(4-15)->mar
Main[mar]->mdr
mdr->acc
End
```

- Definire **storeInd** e **addInd**.

Nuove Istruzioni - 2

- **storeInd**
  - *store indiretta*: addr è l'indirizzo dell'indirizzo della locazione di memoria dove scrivere il contenuto di *accumulatore*.

Listing 5: storeInd

```
ir(4-15)->mar
Main[mar]->mdr
mdr(4-15)->mar
acc->mdr
mdr->Main[mar]
End
```

Nuove Istruzioni - 2

- **addInd**
  - *add indiretta*: addr è l'indirizzo dell'indirizzo della locazione di memoria del valore da sommare ad *accumulatore*.

Listing 6: addInd

```
ir(4-15)->mar
Main[mar]->mdr
mdr(4-15)->mar
Main[mar]->mdr
acc+mdr->mar
End
```

- Salviamo la CPU *Wombat3*.

Wombat3 - Instruction Set

- READ: legge un intero da input e lo mette in ACC
- WRITE: scrive in output il contenuto di ACC
- ADD X: somma il contenuto del registro *acc* alla cella di memoria X e mette il risultato in *acc*.
- SUBTRACT X, MULTIPLY X, DIVIDE X (divisione intera)
- STORE X: da registro ACC alla cella di memoria X
- LOAD X: dalla cella di memoria X al registro ACC
- JUMP X: salta all'istruzione con etichetta X
- JMPN X: salta all'istruzione X se $ACC < 0$
- JMPZ X: salta all'istruzione X se $ACC = 0$
- STOP: segnala la fine del programma
- LOADIND X: dalla cella di memoria indirizzata dal valore contenuto nella cella di memoria X ad *accumulatore*.
- STOREIND X: da *ACC* alla cella di memoria indirizzata dal valore contenuto nella cella di memoria X.
- ADDIND X: somma il contenuto di *ACC* alla cella di memoria indirizzata dal valore contenuto nella cella di memoria X, e pone il risultato in *ACC*.

Esercizio 5

- Utilizzando istruzioni ad indirizzamento indiretto, il programma legge una sequenza di interi e li somma finché non legge un numero negativo. Alla fine stampa la somma (senza includere l'ultimo numero).
- Nota: se definiamo la locazione per la somma ad un indirizzo occupato dal codice del programma, succedono cose strane! (Il codice viene sovrascritto).

Esercizio 5 - Soluzione

Listing 7: Esercizio 5

```
load zero              ;Mem[zero] -> acc
storeInd somma         ;acc -> Mem[Mem[somma]]
inizio: read           ;legge n -> acc
jmpn fine              ;salta a Fine se n < 0.
addInd somma           ;Mem[Mem[somma]] + acc ->acc
storeInd somma         ;acc->Mem[Mem[somma]]
jump inizio            ;legge il prossimo numero
fine: loadInd somma    ;Mem[Mem[somma]]->acc
write                  ;scrive il risultato
stop                   ;si ferma
zero: .data 2 0
somma: .data 2 20      ;locazione (2 byte) per l'ind
                       ;dove memorizzare la somma
```

Esercizio 6

- Utilizzando istruzioni ad indirizzamento indiretto, il programma legge due interi, chiamiamoli *ind* e *cont*. Successivamente, scrive nell'indirizzo di memoria *ind* e nei *cont* successivi i valori $cont, cont - 1 \ldots 1$, rispettivamente.
- E.g. al termine dell'esecuzione del programma con $ind = 20$ e $cont = 4$, la situazione della memoria sarà la seguente.

0	0
2	0
...	...
20	4
22	3
24	2
26	1
...	...

Esercizio 6 - Soluzione

Listing 8: Esercizio 6

```
read ; 1 indirizzo su cui scrivere
store ind ; indirizzo->ind
read ; valore da scrivere nel 1 indirizzo
store cont ; valore->cont

ciclo: jmpz fine
storeInd ind ; acc->indirizzo riferito ind (indiretto)
subtract uno ; decrementa valore
store cont ; aggiorna valore
load ind  ; carica indirizzo
add due ; incrementa indirizzo (2 Byte)
store ind  ; aggiorna indirizzo
load cont ; valore->acc
jump ciclo
fine: stop

ind: .data 2 0  ; locazione base
cont: .data 2 0 ; valore
uno: .data 2 1
due: .data 2 2
```