

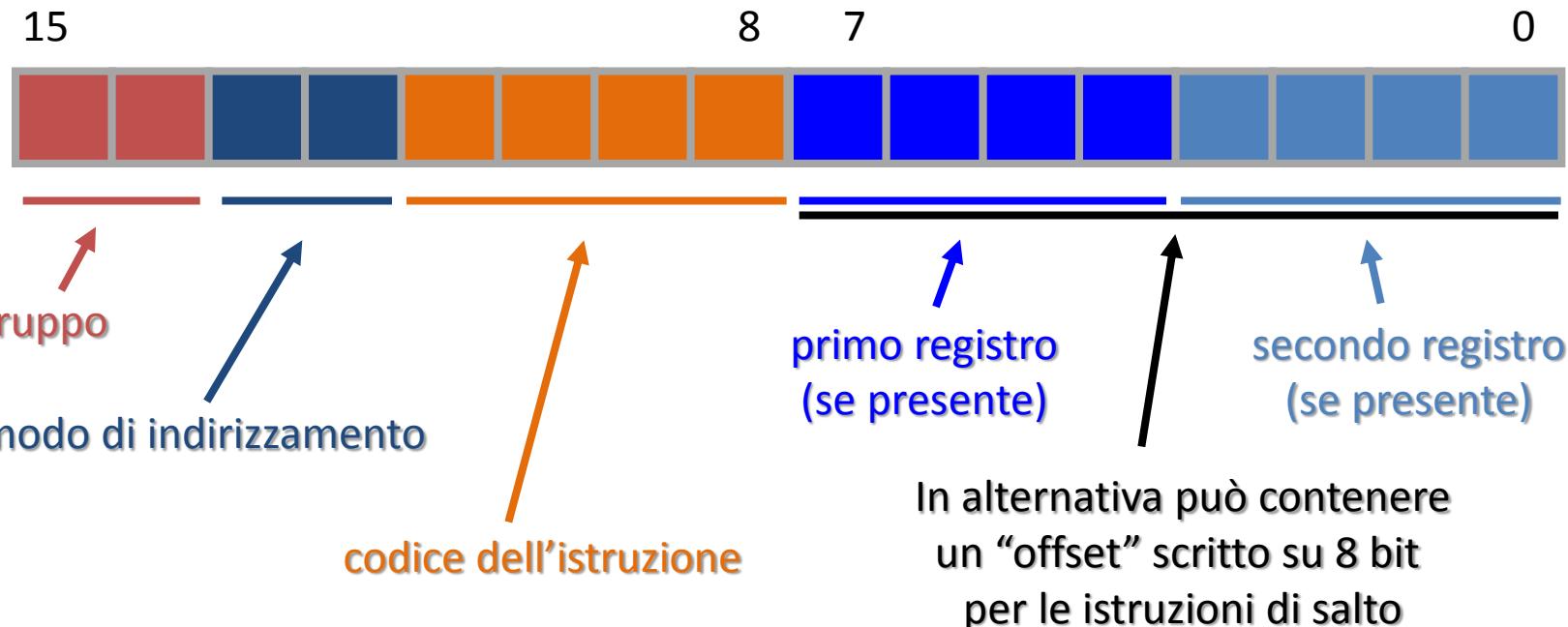
# FONDAMENTI DI INFORMATICA

Prof. PIER LUCA MONTESSORO  
Università degli Studi di Udine

Linguaggio macchina e  
linguaggio assembly



# Formato delle istruzioni



# Gruppi

- Trasferimento dati da e verso la memoria o tra registri  
codice 0 ( $00_2$ )
- Istruzioni aritmetico-logiche  
codice 1 ( $01_2$ )
- Input/output  
codice 2 ( $10_2$ )
- Istruzioni di controllo  
codice 3 ( $11_2$ )



# Modi di indirizzamento

- **Operando immediato**
  - codice 1 ( $01_2$ ), mnemonico I: la parola che segue l'istruzione contiene il dato
- **Indirizzo assoluto**
  - codice 2 ( $10_2$ ), mnemonico A: la parola che segue l'istruzione contiene l'indirizzo del dato
- **Indirizzo in registro**
  - codice 3 ( $11_2$ ), mnemonico R: l'indirizzo del dato è contenuto nel registro specificato



# Gruppo “trasferimento dati”

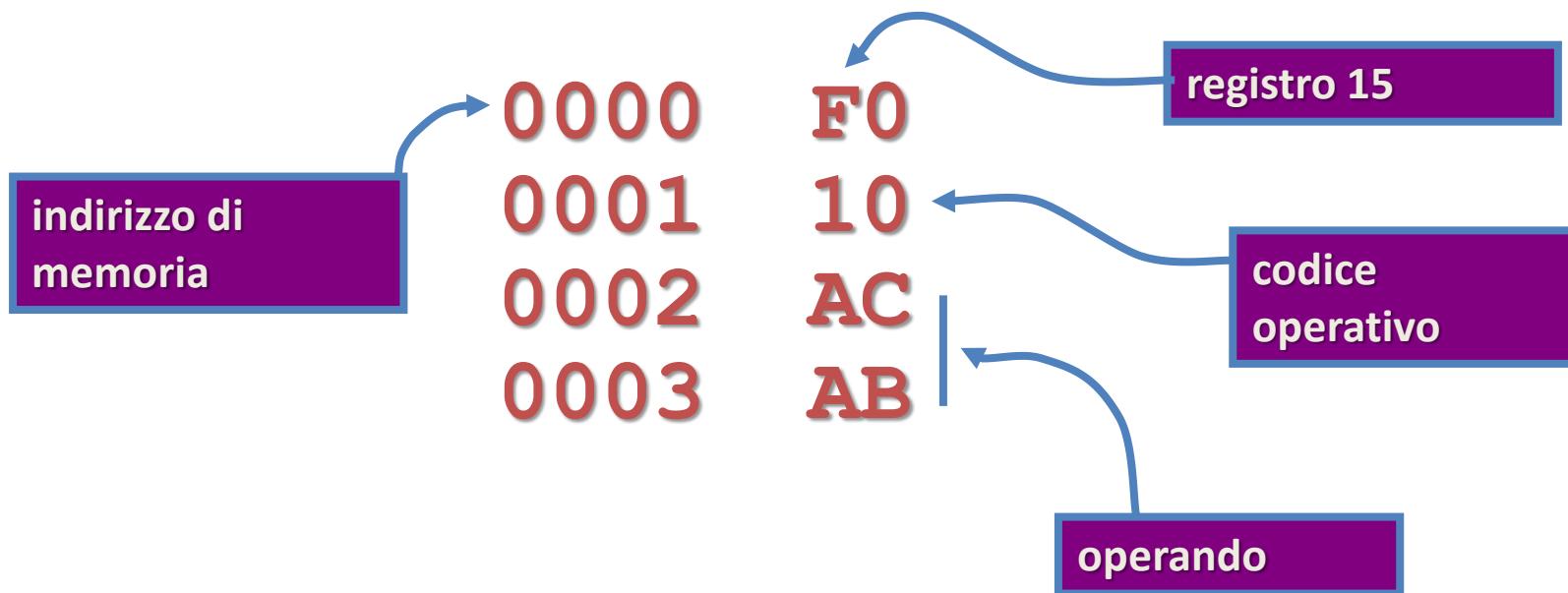
LDWI	d	X	load word	00010000ddd0000	DATA (16)
LDWA	d	A	load word	00100000ddd0000	ADDR (16)
LDWR	d	a	load word	00110000dddaaaa	
LDBI	d	X	load byte	00010001ddd0000	DATA (8)
LDBA	d	A	load byte	00100001ddd0000	ADDR (16)
LDBR	d	a	load byte	00110001dddaaaa	
STWA	s	A	store word	00100010sss0000	ADDR (16)
STWR	s	a	store word	00110010ssssaaa	
STBA	s	A	store byte	00100011sss0000	ADDR (16)
STBR	s	a	store byte	00110011ssssaaa	
MV	s	d	move	00000100ssssddd	
PUSH	s		push	00001000ssss0000	
POP	d		pop	00001001ddd0000	
SPRD	d		read SP	00001101ddd0000	
SPWR	s		write SP	00001110sss0000	

s, d, a: codice del registro interessato  
(s = source, d = destination, a = address)



# Esempio

**LDWI R15 ABAC**



# Gruppo “aritmetico-logiche”

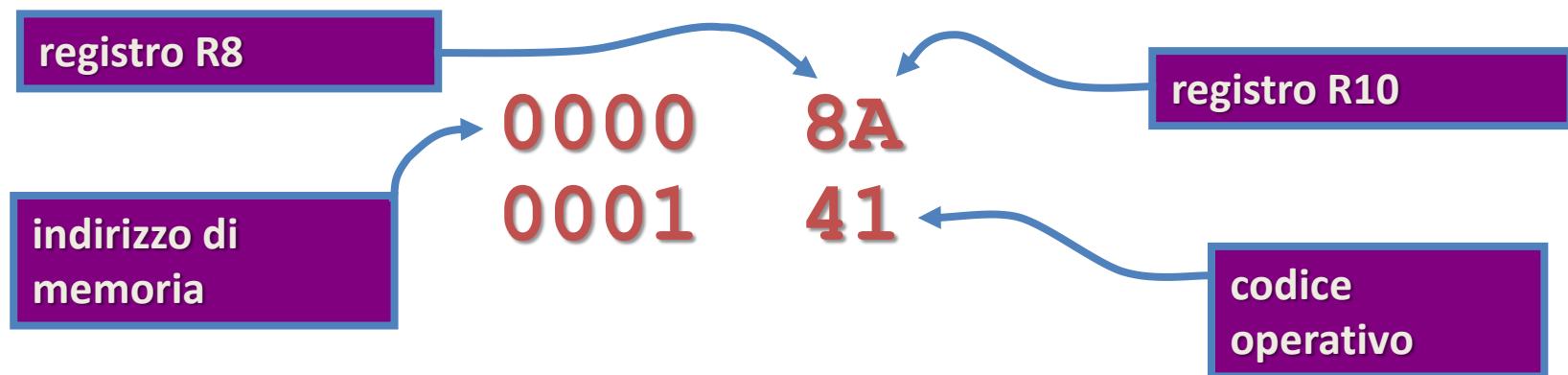
<b>ADD s d</b>	<b>add</b>	<b>01000000ssssddd</b>
<b>SUB s d</b>	<b>subtract</b>	<b>01000001ssssddd</b>
<b>NOT r</b>	<b>bitwise NOT</b>	<b>01000010rrrr0000</b>
<b>AND s d</b>	<b>bitwise AND</b>	<b>01000011ssssddd</b>
<b>OR s d</b>	<b>bitwise OR</b>	<b>01000100ssssddd</b>
<b>XOR s d</b>	<b>bitwise XOR</b>	<b>01000101ssssddd</b>
<b>INC r</b>	<b>increment</b>	<b>01001000rrrr0000</b>
<b>DEC r</b>	<b>decrement</b>	<b>01001001rrrr0000</b>
<b>LSH r</b>	<b>left shift</b>	<b>01001010rrrr0000</b>
<b>RSH r</b>	<b>right shift</b>	<b>01001011rrrr0000</b>

s, d, r: codice del registro interessato  
(s = source, d = destination)



# Esempio

**SUB R8 R10**



# Gruppo “input/output”

INW	d	A	input word	10000000dddd0000	IN_ADDR
INB	d	A	input byte	10000001dddd0000	IN_ADDR
OUTW	s	A	out word	10000010ssss0000	OUT_ADDR
OUTB	s	A	out byte	10000011ssss0000	OUT_ADDR
TSTI	A		test input	1000010000000000	IN_ADDR
TSTO	A		test output	1000010100000000	OUT_ADDR

TSTI e TSTO settano il flag z a 1 (vero) se l'operazione di I/O è stata completata; se il flag vale zero l'operazione di I/O va ripetuta



# Gruppo “controllo”

BR	A	branch	1100000000000000	ADDR
JMP	f	jump	11000001ffffffff	
JMPZ	f	jump if zero	11000010ffffffff	
JMPNZ	f	jump if not zero	11000011ffffffff	
JMPN	f	jump if negative	11000100ffffffff	
JMPNN	f	jump if not neg.	11000101ffffffff	
JMPC	f	jump if carry	11000110ffffffff	
JMPV	f	jump if overflow	11000111ffffffff	
CALL	A	subroutine call	1100100000000000	ADDR
RET		return from sub.	1100100100000000	
HLT		halt	1100111100000000	

ffffffffff = valore dell'offset di salto  
(8 bit in complemento a 2)

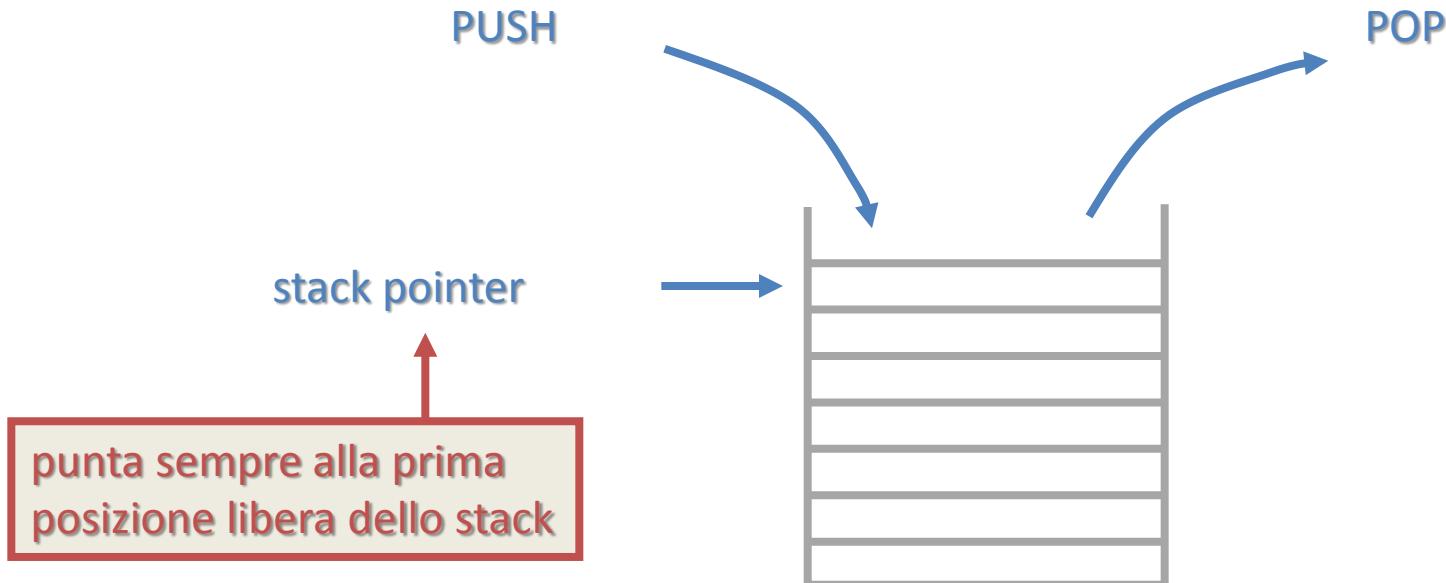


# CALL e RET

- Fanno uso dello stack di sistema (tipo di dato astratto LIFO: Last-In-First-Out implementato mediante vettore)
- Analogia con una pila (“stack”) di piatti: si inserisce in cima alla pila e si preleva in ordine inverso a quello dell’inserimento
- Operazioni
  - PUSH (inserisce)
  - POP (preleva)

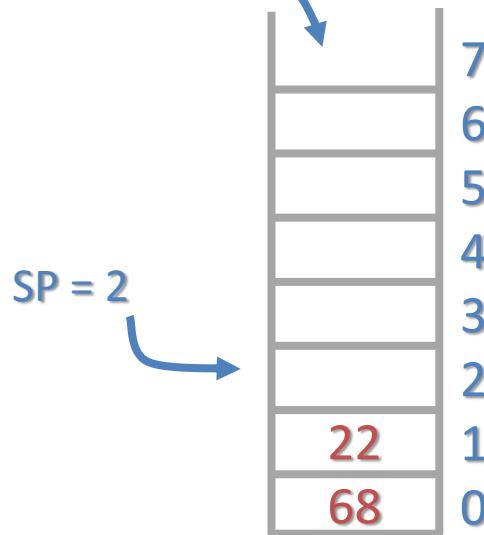


# Stack di sistema

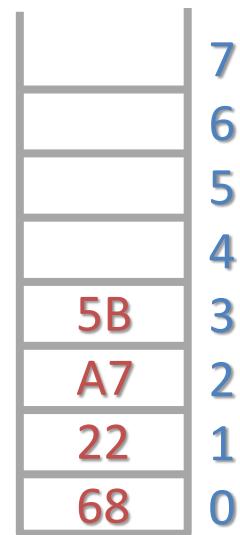


# Esempio

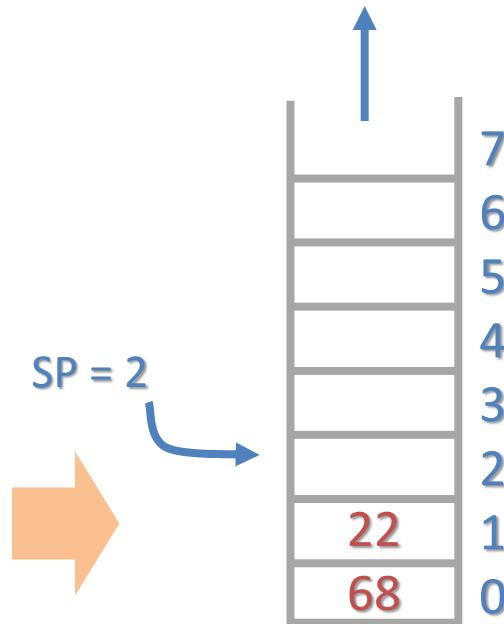
PUSH (A75B)



SP = 4



POP → A75B



# Riepilogo linguaggio

- In the following, uppercase symbols mean numbers, lowercase symbols mean registers. For example, 'A' could be 0x32F5, 'a' could be R12
- 'r', 's', 'd', and 'a' represent the code (0 to 15) of the register to be used:
  - r = register (the value in the register will be read and modified)
  - s = source (read only)
  - d = destination (the value in the register will be overwritten)
  - a = (the value in the register will be used as a memory address)
- The ffffffff value represents a memory offset, positive (forward branches) or negative (in 2's complement, for backward branches)
- Most of the instructions of the data trasfer group work on memory data, addresses or registers:
  - xxxI = data is stored in memory in the location immediately following the instruction code
  - xxxA = the instruction code in memory is followed by the data address
  - xxxR = two registers are used: one to sore the data, and another one to store the memory address



# Riepilogo linguaggio

## Data transfer group:

---

assembly	inst.	name	machine code	action	
LDWI	d X	load word	00010000dddd0000	DATA(16)	d <- X
LDWA	d A	load word	00100000dddd0000	ADDR(16)	d <- mem[A]
LDWR	d a	load word	00110000dddddaaaa		d <- mem[a]
LDBI	d X	load byte	00010001dddd0000	DATA(8)	d <- X
LDBA	d A	load byte	00100001dddd0000	ADDR(16)	d <- mem[A]
LDBR	d a	load byte	00110001dddddaaaa		d <- mem[a]
STWA	s A	store word	00100010sssss0000	ADDR(16)	mem[A] <- s
STWR	s a	store word	00110010ssssaaaa		mem[a] <- s
STBA	s A	store byte	00100011sssss0000	ADDR(16)	mem[A] <- s
STBR	s a	store byte	00110011ssssaaaa		mem[a] <- s
MV	s d	move	00000100ssssddd		d <- s
PUSH	s	push	00001000sssss0000		push (s)
POP	d	pop	00001001dddd0000		d <- pop ()
SPRD	d	read SP	00001101sssss0000		d <- SP
SPWR	s	write SP	00001110sssss0000		SP <- s



# Riepilogo linguaggio

**Arithmetic-logic group:**

---

assembly inst.	name	machine code	action (C operators)
ADD s d	add	01000000ssssddd	d <- d + s
SUB s d	subtract	01000001ssssddd	d <- d - s
NOT r	bitwise NOT	01000010rrrr0000	r <- ~r
AND s d	bitwise AND	01000011ssssddd	d <- d & s
OR s d	bitwise OR	01000100ssssddd	d <- d   s
XOR s d	bitwise XOR	01000101ssssddd	d <- d ^ s
INC r	increment	01001000rrrr0000	r <- r + 1
DEC r	decrement	01001001rrrr0000	r <- r - 1
LSH r	left shift	01001010rrrr0000	r <- r << 1
RSH r	right shift	01001011rrrr0000	r <- r >> 1



# Riepilogo linguaggio

**Input/output group:**

---

assembly inst.	name	machine code	action
INW d A	input word	10000000dddd0000	IN_ADDR(16)
INB d A	input byte	10000001dddd0000	IN_ADDR(16)
OUTW s A	out word	10000010ssss0000	OUT_ADDR(16)
OUTB s A	out byte	10000011ssss0000	OUT_ADDR(16)
TSTI A	test input	1000010000000000	IN_ADDR(16)
			if completed z <- 1 else z <- 0
TSTO A	test output	1000010100000000	OUT_ADDR(16)
			if completed z <- 1 else z <- 0



# Riepilogo linguaggio

Control group:

---

assembly inst.	name	machine code	action (C-like)
BR	A branch	1100000000000000 ADDR(16)	PC <- A
JMP	F jump	11000001FFFFFFFFF	PC <- PC + F
JMPZ	F jump if zero	11000010FFFFFFFFF	if (z == 1) PC <- PC + F
JMPNZ	F jump if not zero	11000011FFFFFFFFF	if (z == 0) PC <- PC + F
JMPN	F jump if negative	11000100FFFFFFFFF	if (N == 1) PC <- PC + F
JMPNN	F jump if not neg.	11000101FFFFFFFFF	if (N == 0) PC <- PC + F
JMPC	F jump if carry	11000110FFFFFFFFF	if (C == 1) PC <- PC + F
JMPV	F jump if overflow	11000111FFFFFFFFF	if (V == 1) PC <- PC + F
CALL	A subroutine call	1100100000000000 ADDR(16)	push (PC); PC <- A
RET	return from sub.	1100100100000000	PC <- pop()
HLT	halt	1100111000000000	halt



# Istruzioni e flag

		Z	N	C	V		Z	N	C	V	
LDWI	d X	load	word	D	D	-	-	LSH	r	left shift	D D D 0
LDWA	d A	load	word	D	D	-	-	RSH	r	right shift	D D D 0
LDWR	d a	load	word	D	D	-	-	INW	d A	input word	D D - -
LDBI	d X	load	byte	D	-	-	-	INB	d A	input byte	D D - -
LDBA	d A	load	byte	D	-	-	-	OUTW	s A	out word	- - - -
LDBR	d a	load	byte	D	-	-	-	OUTB	s A	out byte	- - - -
STWA	s A	store	word	-	-	-	-	TSTI	A	test input	D - - -
STWR	s a	store	word	-	-	-	-	TSTO	A	test output	D - - -
STBA	s A	store	byte	-	-	-	-	BR	A	branch	- - - -
STBR	s a	store	byte	-	-	-	-	JMP	f	jump	- - - -
MV	s d	move		D	D	-	-	JMPZ	f	jump if zero	- - - -
PUSH	s	push		-	-	-	-	JMPNZ	f	jump if not zero	- - - -
POP	d	pop		D	D	-	-	JMPN	f	jump if negative	- - - -
SPRD	d	read	SP	D	D	-	-	JMPNN	f	jump if not neg.	- - - -
SPWR	s	write	SP	-	-	-	-	JMPC	f	jump if carry	- - - -
ADD	s d	add		D	D	D	-	JMPV	f	jump if overflow	- - - -
SUB	s d	subtract		D	D	D	-	CALL	A	subroutine call	- - - -
NOT	r	bitwise	NOT	D	D	0	0	RET		return from sub.	- - - -
AND	s d	bitwise	AND	D	D	0	0	HLT		halt	- - - -
OR	s d	bitwise	OR	D	D	0	0				
XOR	s d	bitwise	XOR	D	D	0	0				
INC	r	increment		D	D	D	D				
DEC	r	decrement		D	D	D	D				

Legenda:

D = dipende dal risultato

- = non modificato

# Assemblatore

- Programma che traduce il programma da linguaggio assembly a linguaggio macchina
- Traduce le istruzioni
- Ignora gli eventuali commenti
- Calcola gli indirizzi di salto
- Riserva spazio in memoria per i dati



Fa uso di “ETICHETTE” e di speciali direttive



# Esempio di programma in assembler

```
; somma N (<= 3) numeri (su 8 bit)
N:    word 3           ; variabile N iniz. a 3
DATA:   byte 5          ; primo numero
        byte 9          ; secondo numero
        byte 0C         ; terzo e ultimo numero
START: LDWA R1 N       ; inizio del programma
       LDWI R2 DATA
       LDWI R10 0      ; azzerà R10
LOOP:   LDBR R11 R2     ; inizia il ciclo
       ADD R11 R10     ; somma al totale
       INC R2          ; passa al num. success.
       DEC R1          ; decrem. il contatore
       JMPNZ LOOP
       HLT              ; in R10 c'e` la somma
```



# Esempio di programma in assembler

```
; somma N (<= 3) numeri (su 8 bit)
N:    word 3           ; variabile N iniz. a 3
DATA:   byte 5          ; primo numero
        byte 9          ; secondo numero
        byte 0C         ; terzo e ultimo numero
START: LDWA R1 N       ; inizio del programma
       LDWI R2 DATA
       LDWI R10 0
LOOP:   LDBR R11 R2
       ADD R11 R10
       INC R2          ; passa ai num. success.
       DEC R1          ; decrem. il contatore
       JMPNZ LOOP
       HLT              ; in R10 c'e` la somma
```

ETICHETTA: rappresenta un indirizzo di memoria il cui valore numerico sarà calcolato dall'assemblatore durante la traduzione del programma in linguaggio macchina



# Esempio di programma in assembler

```
; somma N (<= 3) numeri (su 8 bit)
N:    word 3           ; variabile N iniz. a 3
DATA:   byte 5          ; primo numero
        byte 9          ; secondo numero
        byte 0C          ; terzo e ultimo numero
START: LDWA R1 N       ; inizio del programma
       LDWI R2 DATA
       LDWI R10 0        ; azzerza R10
LOOP:  LDBR R11 R1
       ADD R11 R1
       INC R2
       DEC R1
       JMPNZ LOOP
       HLT                ; in R10 c'e` la somma
```

Direttive per riservare spazio in memoria per le variabili e inizializzarne il valore:  
BYTE: riserva un byte  
WORD: riserva 2 byte



# Esempio di programma in assembler

```
; somma N (<= 3) numeri (su 8 bit)
N:    word 3           ; variabile N iniz. a 3
DATA:   byte 5          ; primo numero
        byte 9          ; secondo numero
        byte 0C         ; terzo e ultimo numero
START: LDWA R1 N       ; inizio del programma
       LDWI R2 DATA
       LDWI R10 0      ; azzerà R10
LOOP:   LDBR R11 R2     ; inizia il ciclo
       ADD R11 R10      ; somma al totale
       LDWI R10 0      ; num. success.
       ADD R11 R10      ; il contatore
```

COMMENTI: l'assemblatore ignora il contenuto della  
riga a partire dal punto e virgola (compreso)

JMPNZ LOOP  
HLT

; in R10 c'e` la somma



# Programma in linguaggio macchina

assembly	indirizzo (h)	contenuto (h)
N: word 3	0000	03
	0001	00
DATA: byte 5	0002	05
byte 9	0003	09
byte 0C	0004	0C
START: LDWA R1 N	0005	10 } 0010000000010000
	0006	20 }
	0007	00 }
	0008	00 }
LDWI R2 DATA	0009	20 }
	000A	10 }
	000B	02 }
	000C	00 }
LDWI R10 0	000D	A0 }
	000E	10 }
	000F	00 }
	0010	00 }
LOOP: LDBR R11 R2	0011	B2 }
	0012	31 }

PRIMA IL  
BYTE MENO  
SIGNIFICATIVO!

segue ➔

# Programma in linguaggio macchina

assembly	indirizzo (h)	contenuto (h)
ADD R11 R10	0013	BA }
	0014	40 }
INC R2	0015	20 }
	0016	48 }
DEC R1	0017	10 }
	0018	49 }
JMPNZ LOOP	0019	F6 }
	001A	C3 }
HALT	001B	00 }
	001C	CF }

Indirizzo della prossima istruzione:  $001B_h$

Indirizzo di destinazione del salto (LOOP):  $0011_h$

Offset del salto:  $0011_h - 001B_h = -000A_h$

$\Rightarrow F6_h$  (in compl. a 2 su 8 bit)



# Esercizio

- Si scriva un programma in linguaggio macchina (in binario) per la lettura di una sequenza di caratteri da tastiera
- Si assuma che:
  - la tastiera abbia indirizzo  $0001_h$
  - la sequenza di caratteri venga terminata dall’utente dal carattere “carriage return”, codice ASCII  $0D_h$ , e in memoria dal valore zero
  - i caratteri letti vadano memorizzati in memoria a partire dall’indirizzo  $0100_h$
  - in  $0000_h$  ci sia il puntatore alla sequenza in memoria ( $0100_h$ )



# Soluzione (assembler)

```
D_ADDR: word 0100          ; indirizzo di memoria dove iniziare  
                           ; a salvare i caratteri letti  
  
START:  LDWA R0 D_ADDR    ; carica l'indirizzo (0100) in R0  
        LDBI R2 0D          ; carica il codice ASCII di CR in R2  
  
LOOP:   INB R1 0001        ; legge un carattere da tastiera  
        TSTI 0001          ; la lettura e` stata eseguita?  
        JMPNZ LOOP         ; no: ripeti l'istruzione di input  
        MV R1 R3            ; salva in R3 il carattere letto  
        SUB R2 R1            ; confronta con il codice ASCII di CR  
        JMPZ END             ; se Z=1 fine della sequenza di input  
        STBR R3 R0            ; salva il carattere letto  
        INC R0               ; incrementa il puntatore  
        JMP LOOP              ; passa al carattere successivo  
        LDBI R3 0              ; carica il valore 0 in R3  
        STBR R3 R0              ; scrive 0 al termine della sequenza  
        HALT
```



# Soluzione (linguaggio macchina)

assembly	indirizzo (h)	contenuto (h)
D_ADDR: word 0100	0000	00 } D_ADDR
	0001	01 }
START: LDWA R0 D_ADDR	0002	00 } 0010000000000000
	0003	20 }
LDBI R2 0D	0004	00 }
	0005	00 } ind. di D_ADDR
LDBI R2 0D	0006	20 } 0001000100100000
	0007	11 }
LOOP: INB R1 0001	0008	0D CR (ASCII 13 <sub>10</sub> )
	0009	10 }
TSTI 0001	000A	81 1000000100010000
	000B	01 }
TSTI 0001	000C	00 } 1: ind. tastiera
	000D	00 }
JMPNZ LOOP	000E	84 1000010000000000
	000F	01 }
JMPNZ LOOP	0010	00 } 1: ind. tastiera
	0011	F6 }
	0012	C3 110000111110110

segue ➔

# Soluzione (linguaggio macchina)

assembly	indirizzo (h)	contenuto (h)
MV R1 R3	0013	13 }
	0014	04 } 0000010000010011
SUB R2 R1	0015	21 }
	0016	41 } 0100000100100001
JMPZ END	0017	06 }
	0018	C2 } 110000100000110
STBR R3 R0	0019	30 }
	001A	33 } 0011001100110000
INC R0	001B	00 }
	001C	48 } 0100100000000000
JMP LOOP	001D	EA }
	001E	C1 } 110000011101010
END:	001F	30 }
	0020	11 } 0001000100110000
	0021	00 valore zero
STBR R3 R0	0022	30 }
	0023	33 } 0011001100110000
HALT	0024	00 }
	0025	CF } 1100111100000000

