Assembly

(9' 07'')

Il programmatore preferisce scrivere le istruzioni usando parole e produce il programma in formato sorgente.

Etichetta	Assembly	Indirizzo	Esa	Cod binario
	MOV A,#0	C009	74H	0111 0100
		C00A	00H	0000 0000
LOOP	INC A	C00B	04H	0000 0100
	INC A	C00C	04H	0000 0100
	LJMP LOOP	C00D	02H	0000 0010
		C00E	C0H	1100 0000
		C00F	0BH	0000 1011

Codice operativo (**opcode**) | Operandi

Campo Etichetta (**label**) | **Campo Istruzione** | **; Campo Commento**

Un'istruzione scritta con 0 e 1 è un'istruzione in linguaggio macchina

Codice sorgente = traduttore => Codice oggetto

Etichetta	Assembly	Indirizzo	Codice oggetto		
	MOV A,#0	C009	74H	00H	
LOOP	INC A	C00B	04H		
	INC B	C00C	04H		
	LJMP LOOP	C00D	02H	C0H	0BH

Quello in blu è un programma sorgente scritto usando l'assembly del µC 8051 e poi tradotto in binario

L'istruzione MOV A,#0 carica il numero 0 nel registro A ed è un'istruzione che occupa 2 Byte, il 2^0 dei quali contiene il numero da caricare in A.

L'istruzione INC A che incrementa A di 1 occupa 1 Byte, perché 1 è già sottinteso in "Incrementa"

L'istruzione LJMP LOOP salta (jump) alla locazione che contiene l'istruzione con etichetta LOOP, ovvero all'indirizzo C00B, e occupa 3 Byte perché l'indirizzo a cui saltare occupa 2 Byte

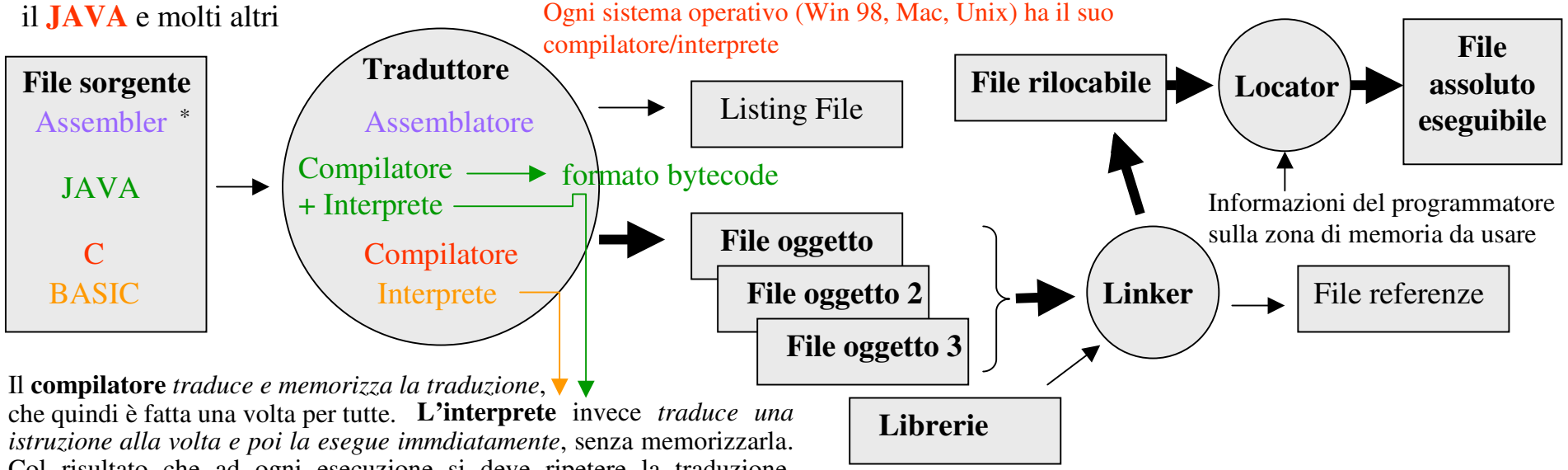
Il linguaggio a basso livello si chiama **Assembler** * ←

E' scomodo per il programmatore e non è portabile, perché ogni µ-processore ha il suo assembler, ma produce i programmi più veloci, usa poca memoria e si usa spesso nei sistemi di controllo

Se una istruzione in linguaggio sorgente dà origine ad una istruzione in linguaggio macchina si parla di linguaggio sorgente **a basso livello.** Se invece una istruzione del sorgente ha bisogno di molte istruzioni in linguaggio macchina per essere eseguita si parla di linguaggio sorgente **ad alto livello**

I linguaggio ad alto livello sono molti: il **C**, il **BASIC** ← il **JAVA** e molti altri

Ogni sistema operativo (Win 98, Mac, Unix) ha il suo compilatore/interprete



Il **compilatore** *traduce e memorizza la traduzione*, che quindi è fatta una volta per tutte. **L'interprete** invece *traduce una istruzione alla volta e poi la esegue immdiatamente*, senza memorizzarla. Col risultato che ad ogni esecuzione si deve ripetere la traduzione, perdendo in velocità

* - Qui è chiamato tutto Assembler, ma più correttamente il linguaggio andrebbe chiamato Assembly, mentre Assembler è il nome del traduttore di un sorgente Assembly

L'assembly dei PIC

(9')

Il set di 35 istruzioni del PIC16CXX e 16FXX

Eseguite quasi tutte in 1 ciclo di 4 Ck ma alcune, come il salto la call e il return richiedono 2 cicli ovvero 8 Ck)

W = accumulatore — F = locaz di RAM, f = quale locaz

	Mnemonic, Operands			Cycles	14-Bit Opcode MSb LSb	Status Affected	Notes
			BYTE-ORIENTED FILE REGISTER OPERATIONS		0 ≤ f ≤ 127		
clrf 05 + clrw + addwf 05, 0 => W = 0	ADDWF	f, d	Add W and f — d=0: W = W + (f) — d=1: (f) = W + (f)	1	00 0111 dfff ffff	C,DC,Z	1,2
	ANDWF	f, d	AND W with f — d=0: W = W and (f) — d=1: (f) = W and (f)	1	00 0101 dfff ffff	Z	1,2
	CLRF	f	Clear f — (05) = 0	1	00 0001 1fff ffff	Z	2
	CLRW	-	Clear W — W = 0	1	00 0001 0xxx xxxx	Z	
	COMF	f, d	Complement f	1	00 1001 dfff ffff	Z	1,2
	DECF	f, d	Decrement f	1	00 0011 dfff ffff	Z	1,2
	DECFSZ	f, d	Decrement f, Skip if 0	1 (2)	00 1011 dfff ffff		1,2,3
	INCF	f, d	Increment f	1	00 1010 dfff ffff	Z	1,2
	INCFSZ	f, d	Increment f, Skip if 0	1 (2)	00 1111 dfff ffff		1,2,3
	IORWF	f, d	Inclusive OR W with f	1	00 0100 dfff ffff	Z	1,2
	MOVF	f, d	Move f	1	00 1000 dfff ffff	Z	1,2
movlw D'10'+movwf H01=>su H01 scrivo D'10'	MOVWF	f	Move W to f — (f) = W — (H01) = W — TMR0 invece di H01	1	00 0000 1fff ffff	---	
	NOP	-	No Operation	1	00 0000 0xx0 0000		
	RLF	f, d	Rotate Left f through Carry	1	00 1101 dfff ffff	C	1,2
	RRF	f, d	Rotate Right f through Carry	1	00 1100 dfff ffff	C	1,2
	SUBWF	f, d	Subtract W from f	1	00 0010 dfff ffff	C,DC,Z	1,2
	SWAPF	f, d	Swap nibbles in f	1	00 1110 dfff ffff		1,2
	XORWF	f, d	Exclusive OR W with f	1	00 0110 dfff ffff	Z	1,2
(permettendo di accedere al banco 0 di RAM)			bit coinvolto (da 0, LSB, a 7, MSB) — **BIT-ORIENTED FILE REGISTER OPERATIONS**				
Cosa fa **bcf status, 5**? Resetta il bit 5 di STATUS	BCF	f, b	Bit Clear f — Azzera il bit b nel registro f o all'indirizzo f	1	01 00bb bfff ffff		1,2
Cosa fa **bsf status, RP0**? Setta il bit 5 di STATUS	BSF	f, b	Bit Set f — Setta (mette a 1) il bit b nel registro f o all'indirizzo f	1	01 01bb bfff ffff		1,2
(permettendo di accedere al banco 1 di RAM)	BTFSC	f, b	Bit Test f, Skip if Clear — Se f(b)=0 salta un'istruzione	1 (2)	01 10bb bfff ffff		3
	BTFSS	f, b	Bit Test f, Skip if Set — Se f(b)=1 salta un'istruzione	1 (2)	01 11bb bfff ffff		3
	Literal = costante		**LITERAL AND CONTROL OPERATIONS** — C =1 se il risultato ha 9 bit — DC = 1 se W > 15				
movlw D'10' + addlw D'12' => W = D'22'	ADDLW	k	Add literal and W — 0 ≤ k ≤ 255 — W = W + k	1	11 111x kkkk kkkk	C,DC,Z	Z=1 se W=0
	ANDLW	k	AND literal with W	1	11 1001 kkkk kkkk	Z	
	CALL	k	Call subroutine	2	10 0kkk kkkk kkkk		
	CLRWDT	-	Clear Watchdog Timer	1	00 0000 0110 0100	$\overline{TO}$,$\overline{PD}$	
Cosa fa ciclo goto ciclo ? Un ciclo infinito	GOTO	k	Go to address — Salta all'indirizzo (o all'etichetta) k	2	10 1kkk kkkk kkkk	$2^{11} = 2^1\,2^{10} = 2x1024$	
	IORLW	k	Inclusive OR literal with W	1	11 1000 kkkk kkkk	Z	
Dopo movlw D'10' in W c'è… D'10' .10	MOVLW	k	Move literal to W — W = k	1	11 00xx kkkk kkkk	---	
	RETFIE	-	Return from interrupt	2	00 0000 0000 1001		
	RETLW	k	Return with literal in W	2	11 01xx kkkk kkkk		
	RETURN	-	Return from Subroutine	2	00 0000 0000 1000		
	SLEEP	-	Go into standby mode	1	00 0000 0110 0011	$\overline{TO}$,$\overline{PD}$	
	SUBLW	k	Subtract W from literal	1	11 110x kkkk kkkk	C,DC,Z	
	XORLW	k	Exclusive OR literal with W	1	11 1010 kkkk kkkk	Z	

Sono istruzione riconosciute dal PIC16F84A ma non dagli altri PIC:

TRIS f dove f=05 (o PORTA) per assegnare W a PORTA o 06 (o PORTB) per assegnare W a portB

OPTION Trasferisce W in OPTION senza bisogno di cambiare banco sui bit RP1 RP0

```
movlw   01010101B   ;Memorizza nel registro
movwf   0CH         ;0CH il valore iniziale da mascherare

movlw   00001111B   ;Prepara la maschera di bit
andwf   0CH,W       ;Effettua l'AND e memorizza il
                    ;risultato nell'accumulatore W
W = 00001111 AND
f = 01010101 =
-----------------
W = 00000101
```

Cosa contiene W se d=W vale d=0?

Memoria di programma