Il linguaggio Assembly

Linguaggio macchina

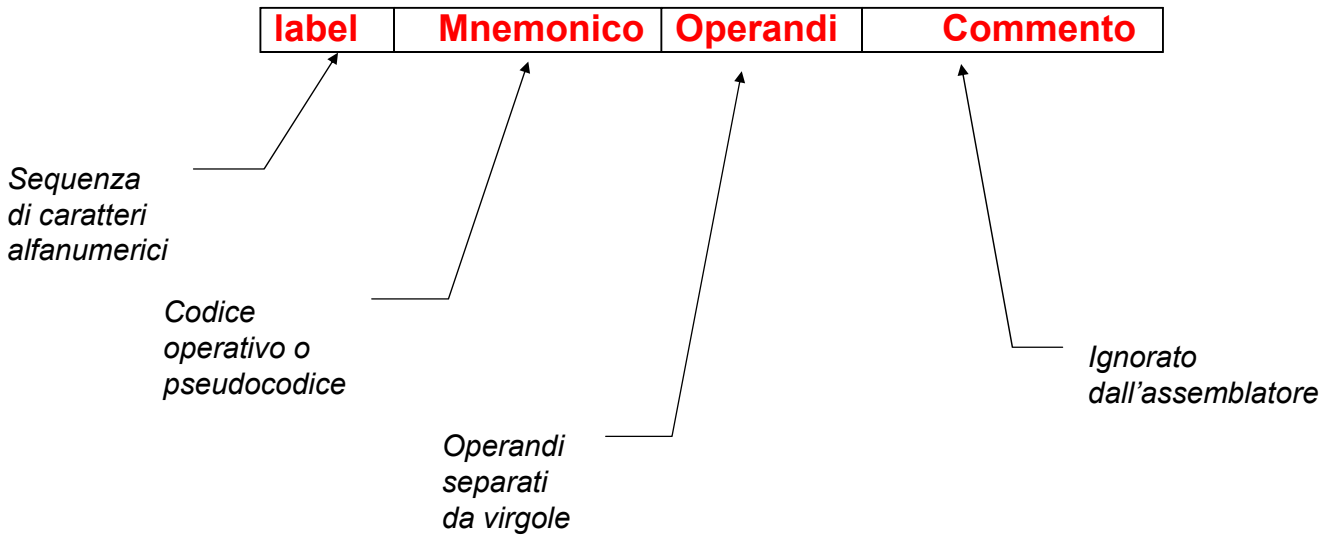
Linguaggio definito da un insieme di istruzioni, codificate come stringhe di bit, che il processore può interpretare ed eseguire direttamente

Linguaggio Assembly

Linguaggio simbolico, vicino al linguaggio macchina, che definisce:

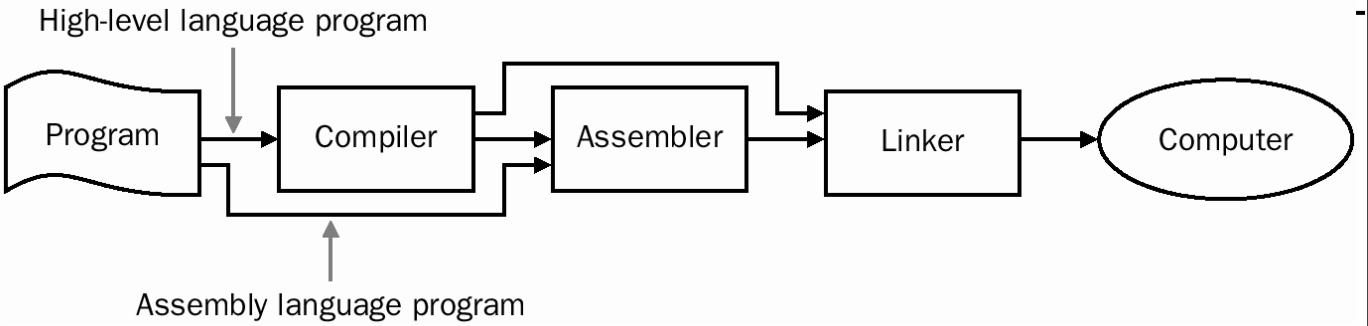
- Uno *mnemonico* per ogni istruzione in L.M.
- Un *formato* per le linee di programma
- Formati per la specifica *della modalità di indirizzamento*
- *Direttive*

Formato istruzioni



Assemblatori

Traduttori linguaggio assembly → linguaggio macchina



Problema dei riferimenti futuri

```
        b label
        ⋮
label   lw ...
```

Soluzione 1 - assemblatori ad 1 passo

La traduzione delle linee di codice contenenti i simboli futuri viene ritardata

Svantaggi

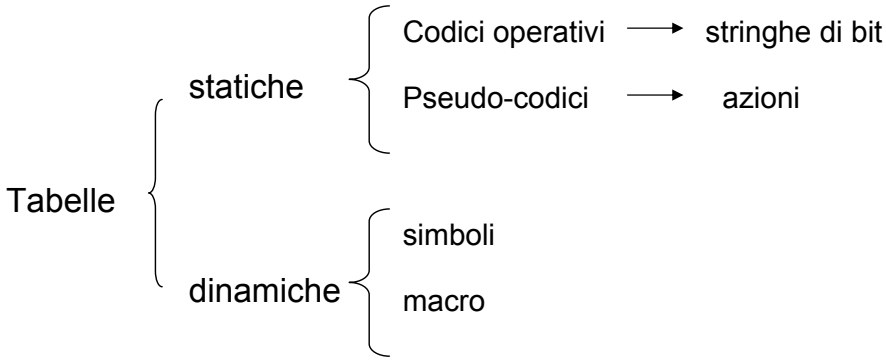
- Grossa richiesta di memoria
- Difficoltà a produrre il listato

- ***Soluzione 2 - assemblatori a 2 passi***

  Nel primo passo viene compilata la tabella dei simboli, nel secondo viene effettuata la traduzione

Tabelle

Vengono utilizzate dall'assemblatore per gestire la corrispondenza simboli ↔ oggetti




```
strlen:    li $v0,0         # Inizializza la lunghezza in $v0
           lw $t0,eol       # Pone in $t0 il terminatore di stringa
loop:      lb $t1,0($a0)    # Carica un carattere in $t1
           beq $t1,$t0,brk  # Esce se è quello di fine stringa
           beqz $t1,brk     # oppure se il byte è vuoto
           addi $v0,$v0,1   # Conteggia un carattere valido
           addi $a0,$a0,1   # Incrementa il puntatore alla stringa
           b loop           # Ripete il ciclo
brk:       jr $ra           # Ritorna al chiamante
```

```
[0x00400064] 0x34020000  ori $2, $0, 0
[0x00400068] 0x3c011001  lui $1, 4097
[0x0040006c] 0x8c280038  lw $8, 56($1)
[0x00400070] 0x80890000  lb $9, 0($4)
[0x00400074] 0x11280000  beq $9, $8, 0
[0x00400078] 0x11200000  beq $9, $0, 0
[0x0040007c] 0x20420001  addi $2, $2, 1
[0x00400080] 0x20840001  addi $4, $4, 1
[0x00400084] 0x0401fffb  bgez $0 -20 [loop-0x00400084]
             brk:        jr $31
```

Istruzioni non completate

Istruzione non ancora tradotta

Simbolo	Valore	Istruzione	
		indirizzo	tipo
strlen	0x400064		
loop	0x400070	0x400084	bgez
brk		0x400074	beq
brk		0x400078	beq

Tabella dei simboli

PLC

E' una variabile interna usata dall'assemblatore per avere traccia degli indirizzi generati per memorizzare istruzioni o dati (Program Location Counter). Per fare riferimento al suo valore in un'istruzione si usa l'asterisco *.

PseudoCodici e Direttive

Forniscono all'assemblatore istruzioni relative all'assemblaggio del programma. La loro interpretazione non genera codice, ma provoca lo svolgimento di particolari azioni da parte dell'assemblatore.

Principali direttive

```
.align n              .half h1,…,hn
.ascii str            .kdata <addr>
.asciiz str           .ktext <addr>
.byte b1,...,bn       .set noat
.data <addr>          .set at
.double d1,…,dn       .space n
.extern sym           .text <addr>
.float f1,…,fn        .word w1,...,wn
.globl sym
```


Costanti Numeriche

12	decimale
$2F	esadecimale

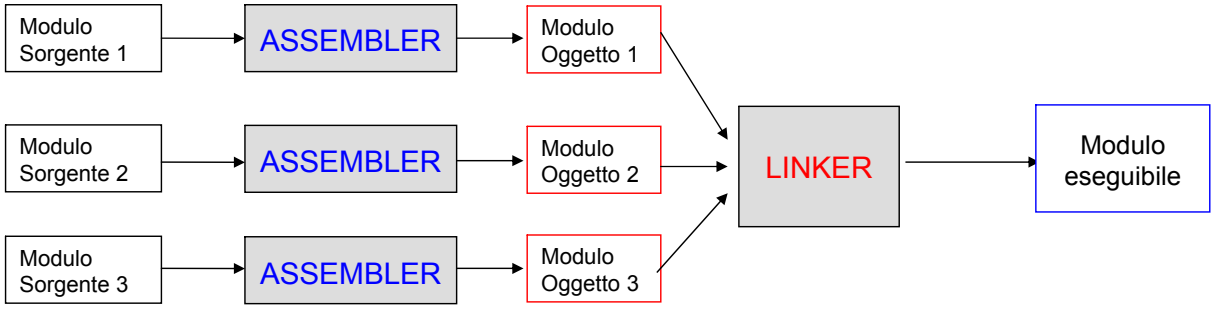
Costanti Carattere

Delimitate da doppi apici. Generano la sequenza di byte corrispondenti ai codici ASCII dei relativi caratteri

```
mesg:  .asciiz  “ciao”
mesg:  .byte    $43,$49,$41,$4F,0
```

Linking (1)

- Divisione del programma in moduli da gestire separatamente
- Il linker permette di collegare più moduli oggetto, assemblati separatamente, in modo da costituire un unico modulo eseguibile finale



Linking (2)

Il modulo oggetto generato dall'assemblatore contiene tutte le informazioni necessarie al linker per operare il collegamento. In particolare, vengono forniti i simboli pubblici (es. procedure esportate dal modulo) ed i riferimenti esterni (es. chiamate a procedure definite in altri moduli).

Identificatore Modulo
Tabella dei simboli pubblici
Tabella dei riferimenti esterni
Codice
Dati
Tabella di rilocazione
Informazioni di debug

Linking (3)

Il linker deve risolvere due problemi:
- rilocazione dei singoli moduli
- risoluzione dei riferimenti esterni

Passi eseguiti dal linker:
1. Costruisce una tabella contenente la lunghezza dei moduli oggetto.
2. In base alla tabella, assegna un indirizzo di caricamento ad ogni modulo oggetto
3. Aggiorna tutti gli indirizzi rilocabili
4. Aggiorna i riferimenti a procedure esterne

Linking (4)

Per realizzare i punti 3 e 4, il linker utilizza le informazioni prodotte dall'assemblatore e contenute nei moduli oggetto.
In particolare, vengono costruite due tabelle generali, che raccolgono i simboli pubblici ed i riferimenti esterni di tutti i moduli.
E' in questa fase che vengono scoperti eventuali errori di collegamento.

tabella dei simboli pubblici

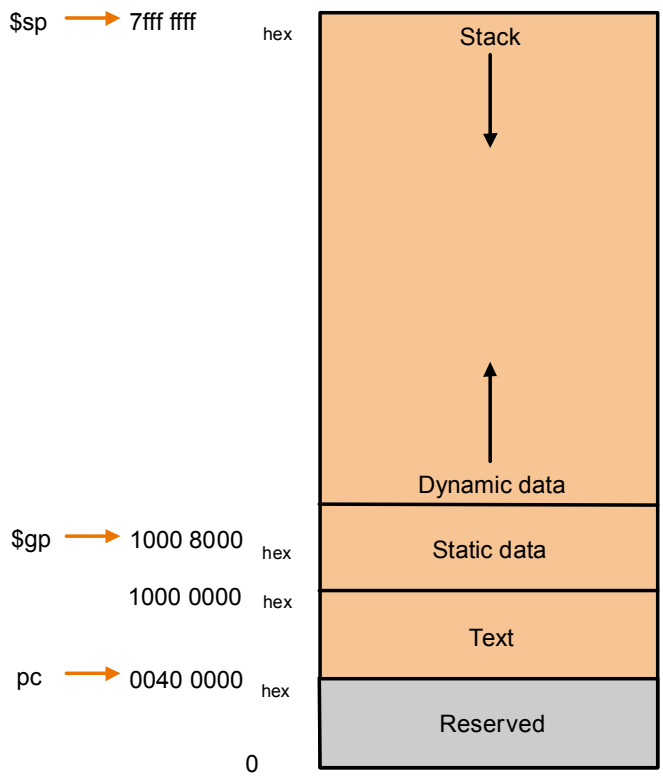
Simbolo pubblico	Modulo	Indirizzo

tabella dei riferimenti esterni

Modulo	Indirizzo	Rif. esterno

In alcuni casi (linking separato dal loading) la rilocazione effettuata non è quella definitiva ed il modulo prodotto è ancora rilocabile.
In questo caso, il linker produce una tabella dei riferimenti rilocabili, memorizzata nel modulo eseguibile e utilizzata dal loader.

Allocazione di memoria del MIPS



Header del file oggetto A

	Nome	Procedura A	
	Dim. del testo	100_{16}	
	Dim. dei dati	20_{16}	
Segmento di testo (codice)	**Indirizzo**	**Istruzione**	
	0	lw $a0, **0**($gp)	
	4	jal **0**	
	...	...	
Segmento dati	**0**	**(X)**	
	...	...	
Informazioni di rilocazione	**Indirizzo**	**Tipo istruzione**	**Dipendenza**
	0	lw	X
	4	jal	B
Tabella dei simboli	**Etichetta**	**Indirizzo**	
	X	-	
	B	-	

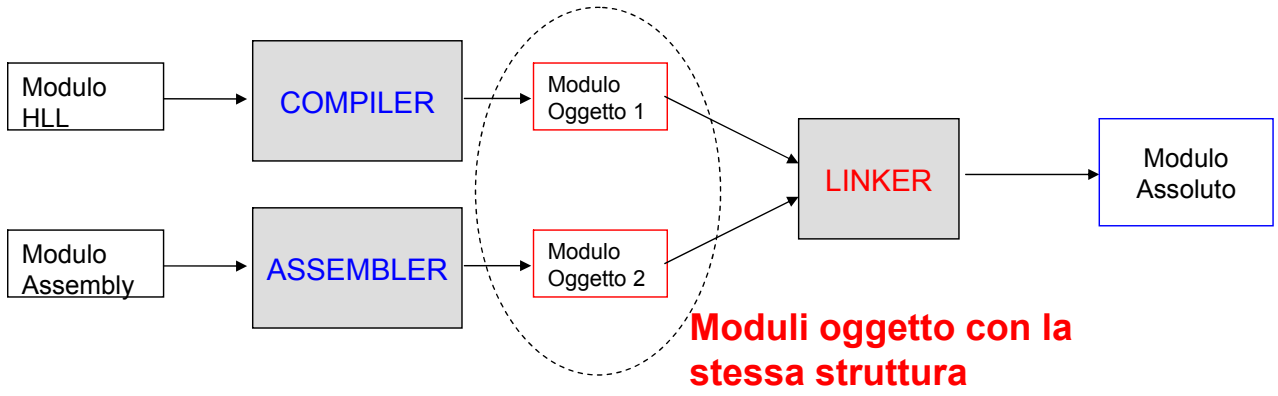
Header del file oggetto B

	Nome	Procedura B	
	Dim. del testo	200_{16}	
	Dim. dei dati	30_{16}	
Segmento di testo (codice)	**Indirizzo**	**Istruzione**	
	0	sw $a1, **0**($gp)	
	4	jal **0**	
	...	...	
Segmento dati	**0**	**(Y)**	
	...	...	
Informazioni di rilocazione	**Indirizzo**	**Tipo istruzione**	**Dipendenza**
	0	lw	Y
	4	jal	A
Tabella dei simboli	**Etichetta**	**Indirizzo**	
	Y	-	
	A	-	

File eseguibile

	Dim. del testo	300_{16}
	Dim. dei dati	50_{16}
Segmento di testo (codice)	**Indirizzo$_{16}$**	**Istruzione**
	0040 0000	lw $a0, **8000**($gp)
	0040 0004	jal **0**
	...	...
	0040 0100	sw $a1, **8020**($gp)
	0040 0104	jal **40 0000**
Segmento dati	1000 0000	**(X)**
	...	...
	1000 0020	**(Y)**
	...	...

Aggancio moduli HLL-moduli assembly



Condizioni necessarie:

- stesse convenzioni per la rappresentazione dei dati
- stesse convenzioni per il subroutine linkage

Cose da sapere

Rappresentazione dei dati

Come sono rappresentati i vari tipi in HLL ?

Subroutine linkage

Relazione nome procedura ⇒ simbolo esterno

Come sono salvati i parametri ?

In che ordine sono salvati i parametri ?

Come viene gestito il valore restituito dalle funzioni ?