

# Laboratorio di Architettura degli Elaboratori

## Fondamenti dei Linguaggi Assembly

Luca Forlizzi, Ph.D.

Versione 18.2



Luca Forlizzi, 2018

© 2018 by Luca Forlizzi. This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/>.

## Caratteristiche degli *ASM*

- Esistono molteplici linguaggi assembly, diversi e incompatibili (o parzialmente compatibili) tra loro
- Tuttavia i concetti fondamentali, la sintassi e la semantica di diversi *ASM* sono abbastanza simili
- Ciò consente di studiarli con una prospettiva generale, ma facendo riferimento ad esempi concreti, il che ci permette di presentare soluzioni diverse

## Caratteristiche degli *ASM*

- Come sappiamo, ogni *ASM* è legato ad un *LM*, in quanto i programmi scritti con un *ASM* vengono di solito tradotti in programmi in uno specifico *LM*
- All'inizio del nostro studio, però, trascureremo questo legame, considerando ciascun *ASM* semplicemente come un linguaggio formale i cui programmi sono eseguiti da una *virtual machine*
- Approfondiremo successivamente i legami con il livello 2
- I linguaggi *ASM* seguono il paradigma imperativo: un programma è una sequenza di comandi, detti *istruzioni*, che effettuano operazioni su dati che sono disponibili in alcuni *dispositivi di memorizzazione*
- I dispositivi di memorizzazione giocano un ruolo analogo a quello che hanno le variabili negli *HLL*

## Caratteristiche degli *ASM*

- Per illustrare gli *ASM-PM* con gli *ASM* associati, faremo riferimento a due esempi specifici
  - MIPS32 Esempio classico di architettura RISC; in particolare faremo uso dell'implementazione MARS
  - M68000 Esempio classico di architettura CISC evoluta; in particolare faremo uso dell'implementazione ASM-One

- MIPS32 è una *ISA* e *ASM-PM* commerciale, progettata da MIPS Technologies Inc. (ex MIPS Computer System Inc.)
- MIPS Technologies Inc. è attualmente una sussidiaria di Imagination Technologies
- MIPS32 e la “sorella” MIPS64, rappresentano la più recente evoluzione della serie di *ISA* e *ASM-PM* denominata MIPS
- MIPS Technologies attualmente si limita a progettare solo le architetture, vendendo ad altre aziende i diritti di realizzare implementazioni
- Chiamiamo CPU MIPS, una CPU che implementa una delle *ISA* MIPS

- La prima CPU MIPS commerciale è MIPS R2000, introdotta nel 1985
- Le CPU MIPS trovano oggi vasto impiego in sistemi embedded quali dispositivi di rete, computer palmari, console per videogame, set-top box
- Negli anni 80 e 90, furono impiegate anche in workstation ad uso scientifico e supercomputer
- Alcuni prodotti commerciali di successo che hanno utilizzato CPU MIPS: workstation SGI, DECstation, Playstation, Nintendo 64, Playstation 2, Playstation Portable

- Le ISA MIPS sono uno degli esempi più classici e rappresentativi della filosofia di progettazione RISC
- Esse sono (relativamente) semplici e lineari e per questo motivo sono molto utilizzate nella didattica, ad esempio nel testo “Struttura e progetto dei calcolatori” [PH5]
- Il Prof. Hennessy, uno degli autori di [PH5], è stato tra i fondatori di MIPS Computer Systems Inc.

# M68000

- M68000 è una *ISA* e *ASM-PM* commerciale, in origine progettata e implementata dalla divisione semiconduttori di Motorola, che oggi è una compagnia indipendente chiamata NXP
- Introdotta nel 1979, insieme alla prima implementazione, la CPU MC68000, è stata una delle prime *ISA* a 32 bit disponibili sul mercato e veniva considerata all'epoca lo stato dell'arte delle *ISA* per CPU a singolo chip
- M68000 è stata impiegata come *ISA* per la famiglia omonima di CPU Motorola e oggi viene usata (in una versione aggiornata chiamata ColdFire) in processori embedded e microcontrollori prodotti da NXP

# M68000

- Le CPU M68000 furono dapprima impiegate, negli anni 80, in workstation basate su Unix
- Successivamente, con il calare dei costi, furono le CPU di molti personal computer degli anni 80 e 90, di stampanti, consolle, palmari e calcolatrici scientifiche
- Alcuni prodotti commerciali di successo che hanno utilizzato CPU M68000

workstation SGI, Sun, Apollo, Next

PC Lisa, Macintosh, Amiga, Atari ST

calcolatrici TI-89, TI-92

palmari Palm Pilot

consolle Sega Mega Drive, Sega Saturn (come coprocessori)

# Struttura di un Programma ASM

- Un programma in un linguaggio ASM è descritto da un testo, chiamato *codice sorgente*, composto da costrutti che soddisfano determinate regole
- Il codice sorgente di un programma è diviso in righe, di solito numerate dall'alto in basso a partire da 1
- I costrutti che compaiono all'interno del codice sorgente di un programma hanno un ordine, detto *ordine testuale*, stabilito come segue:
  - Dati 2 costrutti  $C_1$ ,  $C_2$  che appartengono a due righe diverse,  $C_1$  *precede*  $C_2$  se e solo se la riga a cui appartiene  $C_1$  ha un numero minore di quella a cui appartiene  $C_2$
  - Dati 2 costrutti  $C_1$ ,  $C_2$  che appartengono alla stessa riga,  $C_1$  *precede*  $C_2$  se e solo  $C_1$  si trova più a sinistra di  $C_2$

## Struttura di un Programma *ASM*

- Di norma i programmi *ASM* vengono trasformati in programmi *LM* mediante traduzione
- Un traduttore per un linguaggio *ASM* viene chiamato *assembler* (in italiano *assemblatore*)
- Il processo di traduzione di un programma *ASM* viene detto *assembly process* (in italiano *assemblaggio*)
- Il prodotto della traduzione di un programma *ASM* è un programma *LM* equivalente detto *codice eseguibile* del programma

## Struttura di un Programma *ASM*

- I costrutti presenti nei linguaggi *ASM* si possono di solito dividere nelle seguenti categorie

Definizioni di Label definizioni di simboli che identificano altri costrutti oppure assumono valori costanti

Direttive si tratta di comandi eseguiti durante l'assemblaggio

Istruzioni si tratta di comandi eseguiti a run-time

Commenti testo che viene ignorato dalla *virtual machine*, non ha alcun significato semantico, ma ha lo scopo di fornire informazioni agli esseri umani che leggono il codice sorgente

- Diversamente da quanto accade in moltissimi *HLL*, la sintassi degli *ASM* non permette che un costrutto sia contenuto in un altro costrutto

## Struttura di un Programma *ASM*

- Di solito, ogni riga di un programma *ASM* può contenere al più un solo costrutto per ciascuna delle 4 categorie descritte in precedenza
- Se una riga contiene una definizione di label, essa è il primo costrutto della riga
- Se una riga contiene una direttiva, non contiene un'istruzione e viceversa
- Se una riga contiene un commento, esso è l'ultimo costrutto della riga

## Struttura di un Programma *ASM*

- In molti *ASM*, i programmi vengono divisi in parti chiamate *sezioni* (talvolta *segmenti*)
- L'inizio di una sezione è indicato da una delle direttive apposite, chiamate *direttive di inizio sezione*
- Il termine di una sezione è indicato dalla presenza di una nuova direttiva di inizio sezione, che indica l'inizio di un'altra sezione, oppure dal termine del codice sorgente
- Il contenuto di una sezione è l'insieme dei costrutti compresi tra l'inizio e il termine della sezione

## Struttura di un Programma *ASM*

- L'assemblaggio viene effettuato a partire dal costrutto più a sinistra della riga 1 del codice sorgente, e procede traducendo i costrutti in ordine crescente
- Le definizioni di label vengono memorizzate in tabelle interne all'assembler, per essere usate nella traduzione dei costrutti seguenti
- Le direttive sono comandi per l'assembler, eseguiti durante l'assemblaggio
- Le istruzioni vengono tradotte in istruzioni *LM* che vanno a comporre il codice eseguibile
- I commenti vengono ignorati

# Modi di Funzionamento

- Un dispositivo  $M_4$  definito da un *ASM-PM*, può avere diversi *modi di funzionamento*
- Le principali tipologie di modi di funzionamento sono:
  - OFF  $M_4$  è spento
  - HALT  $M_4$  non esegue istruzioni né operazioni di altro tipo, fino a che qualche evento esterno non modifica il modo di funzionamento
  - TRACE  $M_4$  è bloccato in attesa di un segnale esterno; non appena esso arriva, esegua una singola istruzione e torna a bloccarsi rimanendo nello stesso modo di funzionamento
  - RUN  $M_4$  esegue continuamente istruzioni, in modo sequenziale o parallelo

# Modi di Funzionamento

- Spesso un  $M_4$  è dotato di più modi di funzionamento di tipo RUN, con caratteristiche diverse
- Ciò è particolarmente utile per permettere la realizzazione di sistemi operativi in cui determinate risorse del sistema sono accessibili o meno a seconda del modo di funzionamento in cui si trova il dispositivo
- I modi di funzionamento nei quali un  $M_4$  può utilizzare tutte le sue risorse e capacità vengono chiamati *privileged* o *kernel* o *supervisor*
- I modi di funzionamento nei quali le risorse di un  $M_4$  sono deliberatamente limitate, vengono chiamati *user*
- Alcuni  $M_4$  possiedono diversi modi di tipo user, con caratteristiche e limitazioni differenti

## Esecuzione di un Programma *ASM*

- Nel nostro studio assumiamo che i dispositivi  $M_4$  che eseguono i programmi *ASM* siano *sequenziali*, ovvero eseguano le istruzioni una alla volta (quando si trovano in un modo di funzionamento RUN)
- Chiamiamo *ordine di esecuzione*, l'ordine in cui le istruzioni di un programma vengono eseguite
- Non esiste una regola generale che stabilisce quale sia la prima istruzione di un programma che viene eseguita
- In molti *ASM*, essa è la prima istruzione in ordine testuale di una delle sezioni

## Esecuzione di un Programma ASM

- Quando un  $M_4$  si trova in un modo di funzionamento RUN, dopo aver eseguito una determinata istruzione  $I$ , deve individuare la prossima istruzione da eseguire, che indichiamo come  $\text{Next}(I)$
- Se  $I$  appartiene ad una determinata categoria di istruzioni, chiamate *istruzioni di salto*, quale sia  $\text{Next}(I)$  viene determinato dalla semantica di  $I$
- Altrimenti,  $\text{Next}(I)$  è la successiva di  $I$  in ordine testuale

## Esecuzione di un Programma *ASM*

- In altre parole, negli *ASM* il concetto di esecuzione in sequenza di un insieme di istruzioni viene espresso attraverso la relazione di successione in ordine testuale
- Si tratta di un approccio simile a quello utilizzato nella maggior parte degli *HLL*
- Mostreremo nelle prossime lezioni che la successione delle istruzioni in ordine testuale è legata a come le istruzioni sono memorizzate
- Quindi esiste una relazione tra come le istruzioni sono memorizzate e l'ordine in cui vengono eseguite

- Un'istruzione è un comando per la virtual machine  $M_4$  che indica
  - un'operazione da eseguire
  - gli eventuali dati su cui eseguire l'operazione
  - dove memorizzare l'eventuale risultato calcolato dall'operazione
- Ogni istruzione ha un nome, detto *nome simbolico*
- Salvo rari casi, le operazioni eseguite da una singola istruzione *ASM* sono piuttosto semplici

- Ogni istruzione opera su alcuni dati che vengono chiamati *operandi* dell'istruzione
- In genere, ogni istruzione ha un numero fisso di operandi (mentre istruzioni diverse usano quantità diverse di operandi)
- Ogni istruzione ha vincoli e regole precise su quantità e tipi di operandi ammissibili per essa

- La maggior parte degli *ASM* usano la stessa sintassi generale per le istruzioni  
NOME Specificatore<sub>1</sub>, Specificatore<sub>2</sub>, . . . , Specificatore<sub>N</sub>
- NOME è il nome simbolico
- Specificatore<sub>i</sub> è un'espressione sintattica chiamata *specificatore di operando* che descrive un operando

- Lo stile dei nomi delle istruzioni tende ad essere criptico: si usano di solito nomi di massimo 4 caratteri, che abbreviano espressioni inglesi che indicano la semantica; a volte si abbreviano, senza motivo, anche espressioni che sono già corte
- Esempi di nomi
  - *LEA* (“Load Effective Address”, in M68000 e in *ASM Intel386*)
  - *ROXR* (“Rotate with Extend to Right”, in M68000)
  - *J* (“Jump”, in MIPS)
  - *MOV* (“Move” in *ASM Intel386*)

- Le tipologie più importanti di istruzioni sono:
  - Istruzioni di trasferimento dati
  - Istruzioni aritmetiche
  - Istruzioni logiche e di manipolazione di bit
  - Istruzioni per il controllo del flusso
  - Istruzioni per la gestione della CPU e del sistema

# Operandi

- Un operando può essere un *valore costante*, che fa parte dell'istruzione, oppure un *valore variabile* contenuto in un *dispositivo di memorizzazione*
- In alcune istruzioni, il valore costante o il dispositivo di memorizzazione che contiene un dato variabile sono fissati dalla semantica, ovvero sono sempre gli stessi ogni volta che l'istruzione viene utilizzata in un programma
- Poiché sono fissi, essi non vengono indicati nella sintassi dell'istruzione: per questo sono chiamati operandi *impliciti*
- Gli altri operandi sono invece detti *espliciti* e devono essere indicati dalla sintassi dell'istruzione

# Operandi

- Nei casi in cui un'istruzione calcola dei risultati, li memorizza in uno o più operandi, chiamati *operandi destinazione*
- Gli operandi che non sono operandi destinazione sono detti *operandi sorgente* e non vengono modificati dall'esecuzione dell'istruzione
- Come è ovvio, un operando destinazione non è un valore costante e quindi è contenuto in un dispositivo di memorizzazione

- I dispositivi di memorizzazione che contengono valori variabili, presentano molte similitudini, con le variabili di un *HLL*
- Vi sono però importanti differenze, che evidenzieremo nelle prossime lezioni
- Come le variabili, ogni dispositivo di memorizzazione ha un nome e contiene un valore
- Esiste un concetto di *tipo di dato* per i dispositivi di memorizzazione: però è un po' diverso da quello degli *HLL*

- Esistono due grandi categorie di dispositivi di memorizzazione:
  - ① I *registri*
  - ② Le *parole di memoria*
- Nella maggior parte degli *ASM* troviamo sia registri che parole di memoria

- I registri si differenziano dalle variabili di un *HLL*, in quanto
  - Esiste una quantità fissata di registri, ciascuno con uno o più nomi predefiniti
  - Pertanto i registri non devono essere “dichiarati” in un programma *ASM*: sono automaticamente pronti all’uso
  - Tuttavia i registri non sono inizializzati automaticamente
- Le parole di memoria invece sono molto più simili alle variabili di un *HLL*
  - Non esiste una quantità fissata di parole di memoria
  - È possibile dichiarare parole di memoria, indicandone il nome ed il valore iniziale

- Gli operandi possono quindi essere classificati in
  - Operandi-costante Detti più comunemente *Operandi Immediati* sono valori costanti che possono usati come operandi sorgente in un'operazione; di solito hanno tipo intero
  - Operandi-registro L'operando è un registro che può essere usato come operando sorgente o destinazione
  - Operandi-memoria L'operando è una parola di memoria che può essere usato come operando sorgente o destinazione

# Operandi

- Tipicamente, ogni istruzione ha un numero di operandi espliciti compreso tra 0 e 3
- Le istruzioni a 3 operandi permettono di eseguire un'operazione tra due variabili memorizzando il risultato in una terza variabile come nell'espressione  $UC$

$$V = A \text{ op } B$$

dove  $V$  è una variabile,  $A$  e  $B$  sono ciascuno una variabile oppure una costante e  $op$  è un operatore

# Operandi

- Con 2 operandi (a meno che non siano presenti operandi impliciti) si possono realizzare
  - operazioni tra due variabili che memorizzano il risultato in una delle due, come in  
 $V \text{ as } A$   
dove  $V$  è una variabile,  $A$  è una variabile oppure una costante  
 $\text{as}$  è uno degli operatori di assegnamento composto
  - operazioni unarie su una variabile o costante che memorizzano il risultato in una diversa variabile, come in  
 $V = \text{op } A$   
dove  $V$  è una variabile,  $A$  è una variabile oppure una costante e  
 $\text{op}$  un operatore unario

# Operandi

- Con 1 operando (a meno che non siano presenti operandi impliciti) si possono realizzare operazioni che modificano una variabile, come

$V++$

$V--$

$V = \text{op } V$

dove  $V$  è una variabile e  $\text{op}$  un operatore unario

- Con 0 operandi (a meno che non siano presenti operandi impliciti) si eseguono operazioni che non richiedono dati

- I registri sono dispositivi di memorizzazione cui si accede con uno o più nomi specifici
- Spesso, ma non sempre, sono dispositivi di memorizzazione distinti dalle parole di memoria
- Tipicamente sono i più veloci dispositivi di memorizzazione a disposizione di un  $M_4$  e hanno modalità di accesso specifiche e diverse da quelle delle parole di memoria
- La quantità di registri è molto minore della quantità di parole di memoria

- In relazione al tipo di informazione che memorizzano si classificano in
  - Dati Memorizzano dati da usare nelle operazioni
  - Indirizzi Memorizzano indirizzi di parole di memoria
  - Dati/Indirizzi Memorizzano dati o indirizzi
  - Stato Memorizzano informazioni sullo stato del programma in esecuzione e/o sul funzionamento del dispositivo

# Registri

- In relazione ai loro utilizzi si classificano in
  - Specific purpose Possono essere usati per pochi scopi, spesso per uno solo; di conseguenza sono usati da poche istruzioni
  - General purpose Possono essere usati per scopi differenti, e quindi da molte istruzioni diverse
  - General purpose with special functions Sono *general purpose*, ma hanno alcune particolarità semantiche, spesso in relazione a specifiche istruzioni
- I registri di stato sono molto spesso *specific purpose*
- I registri Dati, Indirizzi e Dati/Indirizzi possono essere *specific purpose* o *general purpose*
- Registri *general purpose* sono più flessibili, ma possono essere più lenti e rendere più complesso il dispositivo

- Registri in MIPS32
  - 32 registri *general purpose* di 32 bit ciascuno (*GPR*)
    - possono contenere indirizzi o dati interi
    - sono numerati da 0 a 31 o indicati con nomi simbolici
    - i registri 0, 1 e 31 hanno anche funzioni speciali
  - 2 registri *specific purpose* di 32 bit ciascuno, chiamati LO e HI, usati per operazioni di moltiplicazione o divisione
  - 32 registri *general purpose* di 32 bit ciascuno per dati floating point
    - il registro *specific purpose* PC
    - vari registri di Stato non interessanti per il nostro corso
- Per maggiori informazioni si rimanda alla documentazione MIPS32

- Registri in M68000
  - 8 registri *general purpose* di 32 bit ciascuno, detti *registri dati*
    - possono contenere dati interi
    - sono chiamati d0,d1,...,d7
  - 8 registri *general purpose* di 32 bit ciascuno, detti *registri indirizzi*
    - possono contenere dati interi o indirizzi
    - sono chiamati a0,a1,...,a7
    - a7 ha funzioni speciali
  - i registri *specific purpose* PC e SR
  - 8 registri *general purpose* di 80 bit ciascuno per dati floating point
  - altri registri di Stato non interessanti per il nostro corso
- Per maggiori informazioni si rimanda alla documentazione M68000

# Istruzioni di trasferimento dati

- Copiano
  - un valore immediato in un dispositivo di memorizzazione
  - oppure il contenuto di un dispositivo di memorizzazione in un secondo dispositivo di memorizzazione
- Sono l'equivalente dell'operazione di assegnamento di un *HLL*
- Alcuni  $M_4$  hanno istruzioni speciali per trasferire sequenze di dati

# Istruzioni di trasferimento dati

- Istruzioni di trasferimento dati in MIPS32
  - In MIPS32 ci sono istruzioni diverse per trasferimenti di tipo diverso
  - Di nostro interesse
    - `move` che copia un dato tra due dei GPR
    - istruzioni che copiano da una parola di memoria ad un GPR
    - istruzioni che copiano da un GPR ad una parola di memoria
    - istruzioni che copiano tra uno dei GPR e HI ed LO
    - `li` che copia un dato immediato in un GPR
    - `la` che copia un indirizzo immediato in un GPR
    - `lui` che copia un valore immediato in una parte di un GPR
  - Per maggiori informazioni su queste ed altre istruzioni di trasferimento si rimanda alla documentazione MIPS32

# Istruzioni di trasferimento dati

- Istruzioni di trasferimento dati in M68000
  - In M68000 c'è una istruzione di trasferimento generale più alcune istruzioni per trasferimenti speciali
  - Di nostro interesse
    - `move` è l'istruzione generale, copia un valore immediato o il contenuto di un registro o parola di memoria in un differente registro o parola di memoria
    - `moveq` istruzione veloce di copia di un valore immediato piccolo (compreso tra -128 e 127) in un registro dati
    - `lea` copia un indirizzo in un registro indirizzi
  - Per maggiori informazioni su queste ed altre istruzioni di trasferimento si rimanda alla documentazione M68000

# Istruzioni aritmetiche

- Si possono classificare in
  - Istruzioni di calcolo di un'operazione aritmetica tra interi
  - Istruzioni di confronto tra interi
  - Istruzioni di calcolo di un'operazione tra floating point
  - Istruzioni di confronto tra floating point
- In LAE ci interessiamo prevalentemente delle prime due categorie
- In questa lezione consideriamo solo la prima categoria

# Istruzioni aritmetiche

- Istruzioni di calcolo di un'operazione aritmetica tra interi in MIPS32
  - La maggior parte sono istruzioni a 3 operandi
    - eseguono un'operazione tra 2 operandi sorgente e memorizzano il risultato nell'operando destinazione
    - uno dei due operandi sorgente è un GPR oppure un valore immediato
    - i restanti due operandi sono GPR
  - Le altre sono istruzioni a 2 operandi
    - gli operandi sono GPR
    - alcune istruzioni per moltiplicazione e divisione usano i registri speciali LO e HI
  - Per maggiori informazioni su queste istruzioni si rimanda alla documentazione MIPS32

# Istruzioni aritmetiche

- Istruzioni di calcolo di un'operazione aritmetica tra interi in M68000
  - Sono quasi tutte istruzioni a 2 operandi
    - eseguono un'operazione tra il valore dell'operando sorgente e quello dell'operando destinazione e memorizzano il risultato nell'operando destinazione
    - entrambi gli operandi possono essere registri dati
    - alcune istruzioni permettono che uno degli operandi sia un valore immediato o una parola di memoria
  - Per maggiori informazioni su queste istruzioni si rimanda alla documentazione M68000

## Direttive di inizio sezione

- Un programma *ASM* può essere diviso in parti chiamate sezioni
- Una sezione può contenere solo istruzioni, oppure solo dati, oppure entrambi
- Vi sono diversi tipi di sezioni per dati, ad esempio sezioni per dati inizializzati e sezioni per dati non inizializzati
- Approfondiremo nelle prossime lezioni

## Direttive di inizio sezione

- Direttive di inizio sezione in MIPS32
  - `.text` è la direttiva di inizio di una sezione che contiene istruzioni
  - `.data` è la direttiva di inizio di una sezione che contiene dati inizializzati
  - entrambe tali direttive possono essere seguite da un numero opzionale che indica l'*indirizzo di inizio sezione*, di cui parleremo nelle prossime lezioni
- Per maggiori informazioni su queste ed altre direttive si rimanda alla documentazione MIPS32

## Direttive di inizio sezione

- Direttive di inizio sezione in M68000
  - `section` è la direttiva di inizio di una sezione
  - può essere seguita da un nome opzionale e da uno specificatore che indica il tipo di sezione
    - `code` indica una sezione che contiene istruzioni
    - `data` indica una sezione che contiene dati inizializzati
    - `bss` indica una sezione che contiene dati non inizializzati
- Per maggiori informazioni su queste ed altre direttive si rimanda alla documentazione M68000

## Direttive di definizione dati

- Le direttive di definizione dati hanno un ruolo simile a quello delle dichiarazioni di variabile di un linguaggio di livello 5
- Mediante una direttiva di definizione dati si possono allocare una o più parole di memoria per memorizzare dei dati
- Si può anche indicare, se si vuole, il valore iniziale di una parola di memoria allocata
- Mediante una label, si può dare un nome ad una parola di memoria e usarla in un programma mediante tale nome
- Approfondiremo nelle prossime lezioni

## Direttive di definizione dati

- Direttive di definizione dati in MIPS32
  - `.word` definisce una o più parole di 32 bit
  - `.half` definisce una o più parole di 16 bit
  - `.byte` definisce una o più parole di 8 bit
  - tali direttive sono seguite da una lista di valori che sono i valori iniziali delle parole di memoria allocate
- Approfondiremo l'argomento nelle prossime lezioni
- Per maggiori informazioni su queste ed altre direttive si rimanda alla documentazione MIPS32

# Direttive di definizione dati

- Direttive di definizione dati in M68000
  - dc definisce una o più parole di memoria
  - tale direttiva è seguita da una lista di valori che sono i valori iniziali delle parole di memoria allocate
- Approfondiremo l'argomento nelle prossime lezioni
- Per maggiori informazioni su queste ed altre direttive si rimanda alla documentazione M68000