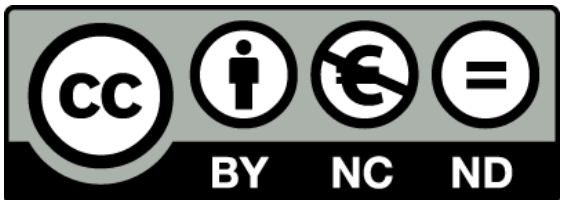
Laboratorio di Programmazione di Sistema

Fondamenti dei Linguaggi Assembly

Luca Forlizzi, Ph.D.

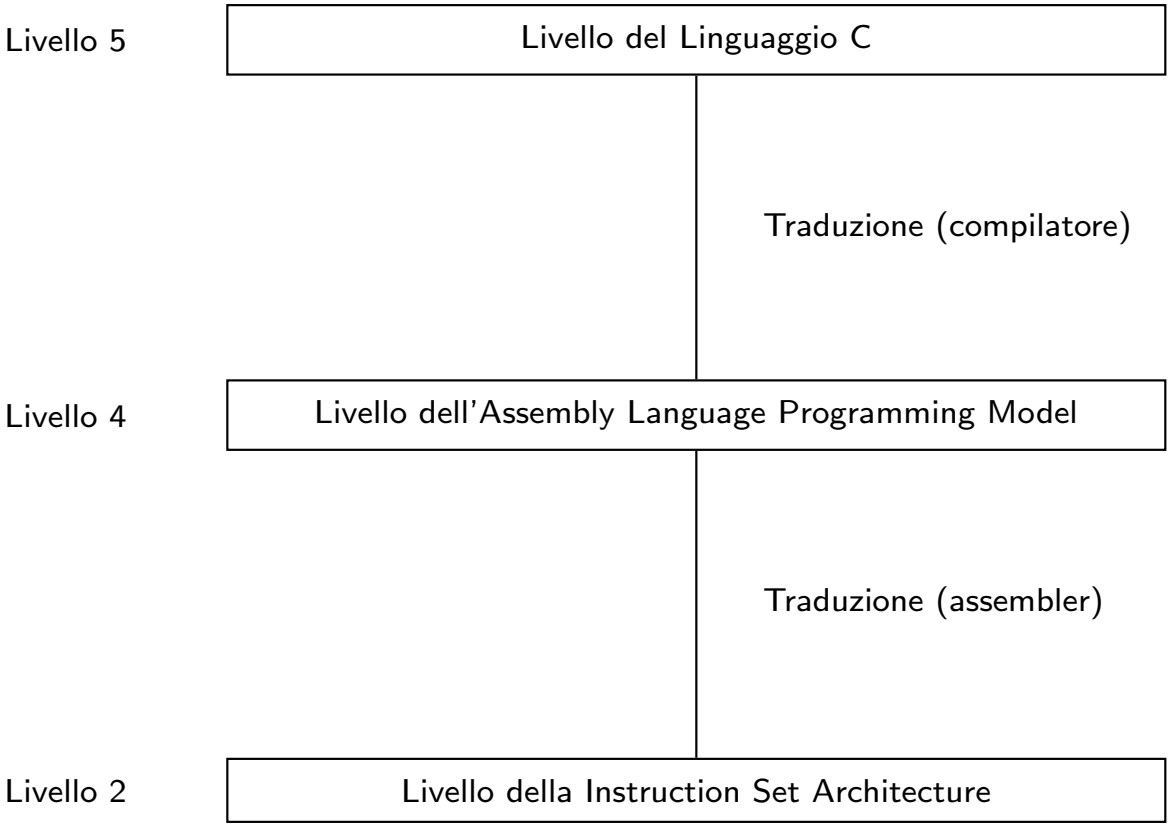
Versione 20.2



Modello a 3 Livelli

- Nelle prime lezioni abbiamo introdotto un modello concettuale di computer come dispositivo di calcolo strutturato in 6 livelli di astrazione
- Tuttavia in LPS non studiamo i livelli della Logica Digitale, della Microarchitettura e del Sistema Operativo
- Faremo quindi riferimento ad un modello di computer semplificato composto da soli 3 livelli di astrazione

Modello a 3 Livelli



Modello a 3 Livelli

- Il nostro interesse primario è studiare quali siano le traduzioni in *LM* di programmi C, e di come esse vengano eseguite da abstract machine di livello 2
- È molto scomodo, non solo scrivere programmi in *LM*, ma anche più semplicemente descrivere come tali linguaggi funzionano, a causa della loro sintassi
- Infatti ciascun costrutto di un *LM* è una stringa binaria, ovvero una sequenza di cifre binarie
- Quindi, per lavorare più comodamente, utilizziamo linguaggi *ASM* al posto dei corrispondenti *LM*

Modello a 3 Livelli

- I linguaggi e i programming model di livello 5, sono relativamente indipendenti da quelli dei livelli sottostanti
- In particolare, un programma scritto in un *HLL* può essere eseguito traducendolo o interpretandolo in molti modi diversi, ovvero mediante linguaggi di livello inferiore completamente differenti gli uni dagli altri
- Al contrario, ogni *ASM-PM A* è *legato* ad una specifica *ISA I*, in quanto l'*ASM* definito dal primo ha una semantica molto simile a quella del *LM* definito dalla seconda
  - I programmi scritti nell'*ASM* descritto da *A* vengono di norma eseguiti traducendoli nel *LM* descritto da *I*
  - L'abstract machine di *A* e quella di *I* hanno struttura e funzionamento molto simili

Modello a 3 Livelli

- Di norma chi realizza una CPU definisce sia la *ISA* che la CPU implementa, sia l'*ASM-PM* legato a tale *ISA*
- Poiché un *ASM-PM* e una *ISA* legati tra loro presentano in genere poche differenze, a parte la sintassi dei rispettivi linguaggi, di solito vengono chiamati con lo stesso nome da chi li definisce
- Tuttavia le implementazioni di una *ISA* e quelle di un *ASM-PM* sono dispositivi concreti ben diversi tra loro
  - Una *ISA* viene implementata in hardware da una CPU
  - Un *ASM-PM* viene implementato da un traduttore (software) che ne trasforma i programmi *ASM* in programmi *LM* equivalenti e da una CPU (hardware)
- Inoltre, per semplicità, i produttori di CPU tendono ad usare lo stesso nome sia per indicare un'architettura che per indicare la abstract machine descritta da tale architettura

Modello a 3 Livelli

- Ciascuna implementazione di un'architettura, può fornire delle *estensioni* al linguaggio definito dall'architettura, ovvero dei costrutti che mettono a disposizione degli utenti servizi aggiuntivi
- Ciò accade molto spesso nel caso degli *ASM-PM*, in quanto essi vengono di solito definiti solo in riferimento alla *ISA* di una CPU, ma le implementazioni degli *ASM-PM* devono operare nel contesto dell'intero computer
- Dunque molte delle estensioni definite da implementazioni di *ASM-PM* sono indispensabili per l'utilizzo pratico delle implementazioni stesse

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 permetterà di presentare modi diversi di definire linguaggi assembly e relativi programming model

Caratteristiche degli *ASM*

- Come detto, 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 *abstract 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*

Modi di Funzionamento

- La abstract machine M definita da un *ASM-PM*, può avere diversi *modi di funzionamento*
- Le principali tipologie di modi di funzionamento sono:

OFF	M è spenta
HALT	M non esegue istruzioni né operazioni di altro tipo, fino a che qualche evento esterno non modifica il modo di funzionamento
TRACE	M è bloccata 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 esegue continuamente istruzioni, in modo sequenziale o parallelo

Modi di Funzionamento

- Spesso una abstract machine M è dotata 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 M può utilizzare tutte le sue risorse e capacità vengono chiamati *privileged* o *kernel* o *supervisor*
- I modi di funzionamento nei quali le risorse di M sono deliberatamente limitate, vengono chiamati *user*
- Alcune abstract machine possiedono diversi modi di tipo user, con caratteristiche e limitazioni differenti

Architetture di Riferimento in LPS

- Per illustrare le caratteristiche degli *ASM* e degli *ASM-PM* associati, faremo riferimento a due esempi specifici

  M68000 Esempio di famiglia di architetture progettate in base alla filosofia CISC

  MIPS Esempio classico di famiglia di architetture progettate in base alla filosofia RISC

- Ciò ci consentirà di mostrare, attraverso esempi reali, soluzioni progettuali differenti
- Per esigenze di sintesi, in LPS verranno omessi molti dettagli delle due architetture
- Per maggiori informazioni
  - Esempi ed esercizi
  - Documentazione ufficiale di M68000 e MIPS32

M68000

- M68000 è il nome di una famiglia di *ISA* e di una famiglia di *ASM-PM* commerciali, in origine progettate e implementate dalla divisione semiconduttori di Motorola, che oggi è una compagnia indipendente chiamata NXP
- La prima architettura della famiglia M68000 è denominata MC68000 e fu introdotta nel 1979, insieme alla sue prime implementazioni
- 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

- La famiglia M68000 non viene più sviluppata; tuttavia molte idee e soluzioni sviluppate con M68000 sono alla base di una nuova famiglia di architetture denominata ColdFire
- Le architetture ColdFire, pur essendo simili a quelle di M68000, hanno, rispetto a esse, un grado di compatibilità basso; proprio per questo vengono considerate una famiglia diversa
- Le CPU che implementano architetture della famiglia ColdFire sono usate in processori embedded e microcontrollori prodotti da NXP

M68000

- Principali versioni di M68000

Anno	Nome	Innovazioni principali nella *ISA*
1979	MC68000	• prima versione di M68000, indirizzi di 24 bit
1982	MC68010	• supporto per la virtualizzazione potenziato
1984	MC68020	• indirizzi di 32 bit • cache per le istruzioni di 256 byte • istruzioni per moltiplicazioni e divisioni con operandi di dimensioni maggiori • istruzioni per gestione bit-field • ulteriori modi di indirizzamento
1987	MC68030	• 2 cache separate per dati e istruzioni da 256 byte ciascuna • unità di gestione della memoria integrata
1990	MC68040	• cache di 4 KB ciascuna • unità floating-point integrata
199?	CPU32	• modulo CPU all'interno dei SOC M68300 • estensione di MC68000 con alcune caratteristiche di MC68020
1994	MC68060	• cache di 8 KB ciascuna • *ISA* superscalare

- In LPS utilizzeremo prevalentemente MC68000, con alcuni riferimenti ad altre versioni

M68000

- Le implementazioni di 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 utilizzano implementazioni di 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)

M68000

- Nell'introduzione della lezione, si è detto che
  - Ci sono poche differenze semantiche tra le architetture e abstract machine di livello 2 e quelle di livello 4 a loro legate, tanto che i produttori di CPU tendono ad usare gli stessi nomi
  - Tuttavia le implementazioni delle architetture di livello 2 e quelle di livello 4 sono dispositivi concreti ben diversi tra loro
- Chiariamo attraverso un esempio relativo ad M68000
- Con M68000 si denota sia una famiglia di *ASM-PM* sia una di *ISA*, che comprendono, tra le altre, l'architettura MC68000
- Con MC68000 si denota sia un *ASM-PM* sia una *ISA*, delle quali esistono numerose implementazioni

M68000

- Un esempio di implementazione della *ISA* MC68000 è la CPU `MC68000P8`
- Il nome di tale CPU ne codifica le caratteristiche principali: oltre alla *ISA* sono indicate la tecnologia microelettronica con cui è realizzata, il range di temperature a cui opera, la forma del contenitore del chip (DIP plastico 64 pin), la frequenza operativa massima (8Mhz), la tensione di alimentazione (5V)
- Si noti che le caratteristiche diverse dalla *ISA*, sono del tutto indipendenti da essa, in quanto non influiscono sul *LM* della CPU; altre implementazioni di MC68000, possono essere diverse da relativamente a queste caratteristiche ma sono comunque compatibili con tale CPU perché eseguono allo stesso modo i programmi

M68000

- Un esempio di implementazione dell'*ASM-PM* MC68000 è invece il software `ASM-One` eseguito su una CPU che implementa la *ISA* MC68000
- `ASM-One` è un traduttore, dotato di IDE, dell'*ASM* di MC68000 disponibile per computer della famiglia Amiga
- I computer Amiga impiegano CPU che implementano una delle *ISA* della famiglia M68000; diversi modelli di Amiga hanno una CPU che implementa MC68000
- Quando sarà necessario utilizzare dettagli del linguaggio *ASM* non definiti dagli *ASM-PM* M68000, faremo riferimento a `ASM-One`
- Indichiamo con il nome MC68000-ASM1, il linguaggio *ASM* definito da MC68000 con le estensioni di `ASM-One`

MIPS

- MIPS è il nome di una famiglia di *ISA* e di una famiglia di *ASM-PM* commerciali, progettate da MIPS Technologies Inc. (ex MIPS Computer System Inc.)
- MIPS Technologies Inc. è attualmente una sussidiaria di Imagination Technologies, e si limita esclusivamente alla progettazione di architetture MIPS, vendendo ad altre aziende i diritti di realizzare implementazioni
- La prima architettura della famiglia MIPS è chiamata MIPS I e fu introdotta nel 1985 assieme alle sue prime implementazioni

MIPS

- 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.

MIPS

- Principali versioni di MIPS

Anno	Nome	Lunghezza indirizzi e registri	Innovazioni e peculiarità principali nella *ISA*
1985	MIPS I	32	• prima versione di MIPS
1989	MIPS II	32	• istruzioni indivisibili di lettura/modifica memoria • predizione dei salti, con relative istruzioni
1991	MIPS III	64	• istruzioni trasferimento dati a 64 bit • istruzioni per operazioni aritmetico-logiche a 64 bit
1994	MIPS IV	64	• unità floating-point migliorata
1999	MIPS32	32	• istruzioni moltiplicazione/divisione a 3 operandi • istruzioni di moltiplicazione-somma
	MIPS64	64	• istruzioni trasferimento dati a 64 bit • istruzioni per operazioni aritmetico-logiche a 64 bit
2002	MIPS32r2	32	• gestione eccezioni potenziata • istruzioni di gestione bit-field
	MIPS64r2	64	
2010	MIPS32r3	32	• istruzioni microMIPS di 16 bit
	MIPS64r3	64	
2014	MIPS32r6	32	• aggiunte/rimosse/modificate istruzioni • scarsa compatibilità con versioni precedenti
	MIPS64r6	64	

- In LPS utilizzeremo prevalentemente MIPS32, con alcuni riferimenti ad altre versioni a 32 bit

MIPS

- Le CPU MIPS sono oggi molto usate in sistemi embedded quali dispositivi di rete, 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 utilizzano implementazioni di MIPS

workstation	SGI Indigo, DECstation 3100, DECstation 5000
supercomputer	NEC Cenju-4, SGI Onyx
embedded	Dreambox (molti modelli), Broadcom SoC
consolle	Playstation, Playstation Portable e Playstation 2, Nintendo 64
automobili	Tesla Model S

MIPS

- Un esempio di implementazione dell'*ASM-PM* MIPS32 è `MARS`
- `MARS` è un simulatore di una semplice *ISA* MIPS32, realizzato in Java
- `MARS` comprende anche un IDE per la programmazione in *ASM*
- Quando sarà necessario utilizzare dettagli del linguaggio *ASM* non definiti dagli *ASM-PM* MIPS, faremo riferimento a `MARS`
- Indichiamo con il nome MIPS32-MARS, il linguaggio *ASM* definito da MIPS32 con le estensioni di `MARS`

Stringhe binarie e bit

- In un *ASM-PM* i dati sono organizzati e manipolati in modo molto simile a come avviene nella corrispondente *ISA*, e vi sono anche similitudini con quanto accade ai livelli della Logica Digitale e della Microarchitettura
- Un *valore binario* o *cifra binaria* è un elemento dell'insieme numerico $\{0, 1\}$
- Ogni dato è formato da una sequenza di valori binari, chiamato *stringa binaria*

Stringhe binarie e bit

- La abstract machine di un *ASM-PM* memorizza dati in *dispositivi di memorizzazione* in essa contenuti
- I dispositivi di memorizzazione più semplici, chiamati *bit* sono in grado di memorizzare una cifra binaria
- I bit sono raggruppati in dispositivi più complessi
  - i *registri*, che contengono di solito da qualche decina a qualche centinaio di bit
  - la *memoria*, che contiene da migliaia a miliardi di bit

Stringhe binarie e bit

- Ogni dato (ovvero ogni stringa binaria) che un programma utilizza durante l'esecuzione deve essere necessariamente memorizzato in un gruppo di bit
- Per ovvi motivi di efficienza, è preferibile che la abstract machine possa leggere, scrivere o modificare un dato mediante una singola operazione
- Ovvero che possa accedere mediante una singola operazione a tutti i bit del gruppo che memorizza il dato
- Le abstract machine possono accedere mediante una singola operazione solo ad alcuni gruppi di bit, chiamati *parole*

Formati di Dato e Parole

- Un *formato di dato* è una parte di un *ASM-PM* (o di una *ISA*) che definisce la struttura e le caratteristiche di un tipo di *parola*, ovvero di un tipo di gruppo di bit a cui la abstract machine è in grado di accedere simultaneamente mediante una singola operazione
- Quindi un formato di dato definisce un insieme di parole
- Il numero di diversi formati di dato di un *ASM-PM* varia, tipicamente, da poche unità fino a una dozzina
- Si noti che in questa lezione descriviamo i formati di dato in relazione agli *ASM-PM*, ma tipicamente un computer presenta gli stessi formati di dato ai livelli 2, 3 e 4

Formati di Dato e Parole

- Per brevità, se p è una delle parole dell'insieme definito dal formato f, diremo che p *appartiene* ad f, o che f *contiene* p
- In dettaglio, un formato di dato stabilisce
  - La *lunghezza* delle parole contenute nel formato, ovvero il numero di bit che formano ciascuna parola che appartiene al formato
  - La *posizione* di ciascun bit all'interno della parola: a ogni bit viene assegnato un numero compreso tra 0 e il valore della lunghezza del formato diminuito di 1, diverso da quello degli altri bit della parola
  - Alcune relazioni tra i bit che lo compongono, tra cui l'*ordine* determinato dalle posizioni
  - Alcuni dettagli delle modalità di accesso

Formati di Dato e Parole

- Un formato di dato non definisce la semantica delle parole, ovvero non attribuisce un significato ai bit che formano le parole
- La semantica delle parole viene definita dall'*interpretazione di dato*, ovvero un algoritmo che, a partire dal valore binario memorizzato in ciascuno dei bit di una parola, calcola un valore, appartenente ad un determinato dominio, detto *valore* della parola
- Negli *ASM-PM*, l'interpretazione di dato è attribuita, ad ogni diverso accesso ad una parola, dall'istruzione che effettua l'accesso, come spiegheremo meglio in seguito

Formati di Dato e Parole

- Tipicamente, una parola è costituita o da bit che si trovano tutti in registri (e in questo caso viene detta *parola di registro*) oppure da bit che si trovano tutti in memoria (in questo caso viene detta *parola di memoria*)
- Più esattamente, rispetto al dispositivo di memorizzazione che ne contiene i bit, distinguiamo 3 categorie di parole
  - *parole di registro*: formate da bit contenuti in uno o più registri della abstract machine
  - *parole di memoria immediate* (*memoria-immediate*): formate da bit della memoria; ciascuna parola memoria-immediata è associata ad una specifica istruzione del programma
  - *parole di memoria ordinarie* (*memoria-ordinarie*): formate da bit della memoria; ciascuna parola memoria-ordinaria ha un *indirizzo* che è necessario indicare per poter accedere alla parola

Formati di Dato e Parole

- Un formato f può contenere parole che appartengono a una, due o tutte e tre le categorie
- Se un formato contiene parole di categorie diverse, può specificare dettagli diversi per ciascuna categoria in merito all'organizzazione dei bit
- La lunghezza del formato è comunque la stessa per tutte le parole, indipendentemente dalla categoria
- Nelle prime lezioni di LPS i dati verranno memorizzati solo in parole di registro e in parole memoria-immediate

Parole di Registro

- I registri sono dispositivi di memorizzazione cui si accede con uno o più nomi specifici
- Tipicamente sono i più veloci dispositivi di memorizzazione di una abstract machine e hanno modalità di accesso specifiche e diverse da quelle della memoria
- La quantità di bit presente in tutti i registri è molto minore della quantità di bit contenuta dalla memoria

Parole di Registro

- In relazione al tipo di informazione che memorizzano, i registri si classificano in

Dati	Memorizzano dati da usare nelle operazioni
Indirizzi	Memorizzano indirizzi di memoria
Dati/Indirizzi	Memorizzano dati o indirizzi
Stato	Memorizzano informazioni sullo stato del programma in esecuzione e/o sul funzionamento del dispositivo

Parole di Registro

- 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, 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

Parole di Registro

- Registri in MC68000
  - 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` (di 32 bit) e `sr` (di 16 bit)
  - come opzione, 8 registri *general purpose* di 80 bit ciascuno per dati floating point
  - altri registri di Stato non interessanti per il nostro corso

Parole di Registro

- Registri in MIPS32
  - 32 registri *general purpose* di 32 bit ciascuno, detti *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

Parole di Registro

- In ogni *ASM-PM*, l'intero contenuto di un registro è una parola
- Ovvero, per ciascun registro esiste un formato di dato che definisce come parola di registro l'intero contenuto del registro
- Può accadere che in un *ASM-PM* ci siano registri di lunghezze diverse: per ciascuna possibile lunghezza esiste un corrispondente formato di dato
  - Ad esempio in MC68000, `sr` è di 16 bit, mentre tutti gli altri registri sono lunghi 32 bit: esistono quindi i corrispondenti formati di dato

Parole di Registro

- In aggiunta alle parole di registro che coincidono con l'intero contenuto di un registro, ce ne possono essere altre che coincidono con alcune parti di un registro
- In molti *ASM-PM*, detto f un formato di lunghezza N che definisce una parola coincidente con un intero registro R, esistono dei formati $f', f'', \ldots$ di lunghezze (rispettivamente) $\frac{N}{2}, \frac{N}{4}, \ldots$ che definiscono parole formate da bit di R che hanno posizioni consecutive (facendo riferimento alle posizioni stabilite da f)
- Un formato di dato che si riferisce a parole di registro, può riferirsi anche a parole di altre categorie (con la stessa lunghezza)

Parole memoria-immediate

- Le parole memoria-immediate vengono impiegate per memorizzare dati cui deve accedere, prevalentemente, una singola istruzione
- L'istruzione associata ad una parola memoria-immediata, infatti, può accedere al contenuto di tale parola in modo molto efficiente e senza bisogno dell'indirizzo della parola
- Al contrario, le altre istruzioni non sono associate alla parola e quindi, di norma, non possono accedere ad essa
  - In molti *ASM-PM* è del tutto impossibile che un'istruzione possa accedere ad una parola memoria-immediata non associata all'istruzione stessa
  - A livello di *ISA* può essere possibile tale accesso, ma richiede tecniche speciali e non è efficiente
- Le parole memoria-immediate vengono utilizzate in prevalenza per memorizzare dati non modificabili durante l'esecuzione del programma, ovvero costanti

Formati di Dato Generali e Speciali

- È utile classificare i formati di dato rispetto alla quantità di istruzioni che li usano
  - Chiamiamo *generali* i formati di dato che vengono usati da molte istruzioni *ASM*
  - Al contrario, chiamiamo *speciali* i formati di dato utilizzati da poche istruzioni
- Spesso un formato di dato di dato speciale contiene parole di una sola categoria (ad esempio solo parole registro oppure solo parole memoria-ordinarie)
- Di solito un formato di dato generale contiene parole di più di una categoria

Formato e Interpretazione di Dato

- Come detto, un formato di dato non stabilisce l'interpretazione di dato; essa viene invece stabilita da un'istruzione che accede ad un dato
- Nella maggior parte degli *ASM-PM*, ogni istruzione, per ciascun dato
  - Usa un'unica interpretazione di dato
  - Può usare diversi formati di dato
- Un formato di dato può essere associato (da istruzioni diverse) a diverse interpretazioni di dato, ma non necessariamente a tutte quelle possibili nell'*ASM-PM*
- Un'interpretazione di dato viene associata ad uno specifico formato di dato, tuttavia a formati differenti possono essere associate interpretazioni simili

Formato e Interpretazione di Dato

- Le interpretazioni di dato consentono di utilizzare gruppi di cifre binarie per rappresentare dati di varia natura
- Le principali interpretazioni di dato, presenti nella maggior parte degli *ASM-PM*, permettono di rappresentare
  - Numeri interi con un numero fissato di cifre
  - Indirizzi di memoria
  - Numeri floating point
- Le interpretazioni di dato per numeri interi (con numero fissato di cifre) e indirizzi di memoria verrano studiate in future lezioni di LPS
- Le interpretazioni di dato per numeri floating point sono al di fuori dell'ambito di LPS

Dati: *HLL* vs. *ASM*

- Come sappiamo, nei linguaggi ad alto livello imperativi, ciascun dato ha associato un *tipo di dato*
- Il tipo di dato di una costante o di una variabile ne specifica
  - L'aspetto sintattico, ovvero la quantità di bit usati per formare la variabile e la loro organizzazione
  - L'aspetto semantico, ovvero l'algoritmo che, a partire dal valore assunto da ciascuna delle cifre binarie del dato, permette di calcolare un valore, appartenente ad un determinato dominio

Dati: *HLL* vs. *ASM*

- Prima differenza concettuale sui dati tra gli *HLL* e gli *ASM*
  - Negli *HLL* gli aspetti sintattici e semantici dei dati sono integrati nel concetto di tipo di dato
  - Negli *ASM*, il formato di dato (che definisce gli aspetti sintattici) e l'interpretazione di dato (aspetti semantici) sono concetti distinti, benché ovviamente collegati

Dati: *HLL* vs. *ASM*

- Seconda differenza fondamentale tra gli *HLL* e gli *ASM*
  - Negli *HLL* il tipo di dato è una proprietà di un dato, che determina quali operazioni sono valide o meno su di esso
  - Negli *ASM* formato ed interpretazione di un dato dipendono in parte dalla parola che lo memorizza e in parte dalle operazioni che vengono effettuate
- In particolare, negli *HLL* avviene il *type checking* che determina quali operazioni sono consentite su un dato
- Invece negli *ASM* non vi è *type checking* a limitare quali sono i dati a cui può essere applicata una determinata operazione

Dati: *HLL* vs. *ASM*

- Le parole di registro si differenziano dalle variabili di un *HLL*, in quanto
  - Esiste una quantità fissata di registri e di parole di registro, ciascuno con uno o più nomi predefiniti
  - Pertanto le parole di registro non devono essere "dichiarate" in un programma *ASM*: sono automaticamente pronti all'uso
  - Tuttavia le parole di registro non sono inizializzate automaticamente
- Le parole di memoria sono più simili alle variabili di un *HLL*
  - Non esiste una quantità fissata di parole di memoria
  - Esistono costrutti *ASM* analoghi alle dichiarazioni di variabile, per indicare il nome ed il contenuto iniziale di una parola di memoria

Formati di Dato Generali in MC68000

- MC68000 ha 3 formati di dato generali
  - `long` di lunghezza 32
  - `word` di lunghezza 16
  - `byte` di lunghezza 8
- Ciascuno di questi 3 formati definisce sia parole di registro, sia parole memoria-ordinarie, sia parole memoria-immediate
- Descriviamo brevemente le caratteristiche di tali formati in riferimento a parole di registro e parole memoria-immediate, quelle relative a parole memoria-ordinarie verranno studiate successivamente
- Caratteristiche del formato `long`
  - lunghezza 32
  - definisce come parole l'intero contenuto di tutti i registri dati e i registri indirizzi

Formati di Dato Generali in MC68000

- Caratteristiche del formato `word`
  - lunghezza 16
  - per ciascun registro dati o indirizzi, definisce la parola formata dai bit che in `long` hanno posizione compresa tra 0 e 15
  - ai bit dei registri dati e indirizzi contenuti sia in un parola di `long` che in una di `word`, vengono assegnate le stesse posizioni
  - definisce come parola l'intero contenuto di `sr`
- Caratteristiche del formato `byte`
  - lunghezza 8
  - è un formato non valido per i registri indirizzi
  - per ciascun registro dati e per `sr`, definisce la parola formata dai bit che in `word` hanno posizione compresa tra 0 e 7
  - ai bit dei registri dati e e di `sr` contenuti sia in un parola di `byte` che in una di `word`, vengono assegnate le stesse posizioni

Formati di Dato Generali in MIPS32

- MIPS32 ha 2 formati di dato generali
  - `word` di lunghezza 32
  - `half` di lunghezza 16
- Descriviamo brevemente le caratteristiche di tali formati in riferimento a parole di registro e parole memoria-immediate, quelle relative a parole memoria-ordinarie verranno studiate successivamente

Formati di Dato Generali in MIPS32

- Caratteristiche del formato `word`
  - lunghezza 32
  - definisce come parole l'intero contenuto di ciascuno dei registri
- Caratteristiche del formato `half`
  - lunghezza 16
  - è un formato non valido per registri
- Si osservi che l'unico formato di dato generale valido per i registri è `word`

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 *abstract 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*

- 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

Sezioni

- Un programma *ASM* può essere diviso in parti chiamate *sezioni* (talvolta *segmenti*)
- Una sezione può contenere solo istruzioni, oppure solo dati (in memoria), oppure entrambi
- Vi sono diversi tipi di sezioni per dati, ad esempio sezioni per dati inizializzati e sezioni per dati non inizializzati

Sezioni

- L'inizio di una sezione è indicato da una delle direttive apposite, chiamate *direttive di inizio sezione*
- Il termine di una sezione è indicato in uno dei seguenti modi:
  - dalla presenza di una nuova direttiva di inizio sezione, che indica l'inizio di un'altra sezione
  - dalla presenza di una direttiva di fine programma, che indica che non vi sono più costrutti da tradurre
  - dal termine del codice sorgente
- Il contenuto di una sezione è l'insieme dei costrutti compresi tra l'inizio e il termine della sezione

Sezioni

- Alcune direttive di MC68000-ASM1 per la strutturazione del programma
  - `org` è una direttiva di inizio sezione; è seguita da un valore numerico che indica l'*indirizzo di inizio sezione*, di cui parleremo nelle prossime lezioni
  - `section` è una direttiva di inizio 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 direttive per definire parole di memoria inizializzate
    - `bss` indica una sezione che contiene direttive per definire parole di memoria non inizializzate
  - `end` indica il termine del programma; qualsiasi cosa sia scritta nel codice sorgente dopo la direttiva `end` viene considerata un commento

Sezioni

- Alcune direttive di MIPS32 per la strutturazione del programma
  - `.text` indica l'inizio di una sezione che contiene istruzioni
  - `.data` indica l'inizio di una sezione che contiene direttive per definire parole di memoria inizializzate
  - entrambe tali direttive possono essere seguite da un numero opzionale che indica l'*indirizzo di inizio sezione*, di cui parleremo nelle prossime lezioni

Esecuzione di un Programma *ASM*

- In LPS ci limitiamo a considerare abstract machine che eseguono i programmi *ASM* in modo *sequenziale*, ovvero che eseguono 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 una abstract machine 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 $\mathrm{Next}(I)$
- Se *I* appartiene ad una determinata categoria di istruzioni, chiamate *istruzioni di salto*, quale sia $\mathrm{Next}(I)$ viene determinato dalla semantica di *I*
- Altrimenti, $\mathrm{Next}(I)$ 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

Istruzioni

- Un'istruzione è un comando per la abstract machine che specifica
  - un'operazione da eseguire
  - gli eventuali dati su cui eseguire l'operazione; in particolare per ciascun dato
    - come accedere ad esso
    - quale formato di dato usare
    - quale interpretazione di dato usare
  - come e in quali dispositivi 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

Istruzioni

- 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)

Istruzioni

- 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

- Come sappiamo, le istruzioni effettuano delle operazioni su dei dati e producono un risultato
- Quindi per eseguire un'istruzione, la abstract machine deve
  - *leggere* i dati, accedendo alle parole che li memorizzano
  - *scrivere* il risultato in una parola di memorizzazione
- Le parole da cui un'istruzione legge i sui dati e la parola in cui essa memorizza il suo risultato, sono chiamati *operandi* dell'istruzione

Operandi

- In particolare, le parole memoria-immediate, sono chiamati *operandi immediati*
- Nella maggior parte degli *ASM-PM*, si ha che
  - L'istruzione associata ad una parola memoria-immediata è anche l'unica istruzione che accede a tale parola
  - Di conseguenza è utile memorizzare in una parola memoria-immediata dati che servono all'istruzione associata alla parola, ma non dati che devono essere usati da altre istruzioni
  - Quindi gli operandi immediati non possono essere usati per memorizzare il risultato di un'istruzione, perché il risultato di un'istruzione deve servire come dato alle istruzioni successive

Operandi

- Ogni istruzione ha vincoli e regole precise su
  - quantità di operandi ammissibili (in quasi tutti i casi il numero di operandi è fissato)
  - formati di dato ammissibili
  - modi di specificare quali sono gli operandi

Operandi

- In alcune istruzioni, uno o più operandi sono fissati dalla semantica, ovvero sono sempre gli stessi ogni volta che l'istruzione viene utilizzata in un programma
- Gli operandi fissati dalla semantica, non vengono indicati nella sintassi dell'istruzione (in quanto non è necessario): per questo sono chiamati operandi *impliciti*
- Gli altri operandi sono invece detti *espliciti* e devono essere indicati dalla sintassi dell'istruzione

Operandi

- La maggior parte degli *ASM* usano la stessa sintassi generale per le istruzioni
  NOME $\mathrm{Specificatore}_1$, $\mathrm{Specificatore}_2$, ..., $\mathrm{Specificatore}_N$
- NOME è il nome simbolico
- $\mathrm{Specificatore}_i$ è un'espressione sintattica chiamata *specificatore di operando* che descrive un operando esplicito

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 op1 B`
  dove `V` è una variabile, `A` e `B` sono ciascuno una variabile oppure una costante e `op1` 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 op2 A`
    dove `V` è una variabile, `A` è una variabile oppure una costante, op2 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 = op3 A`
    dove `V` è una variabile, `A` è una variabile oppure una costante e op3 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 = op4 V`
  dove `V` è una variabile e `op4` un operatore unario
- Con 0 operandi (a meno che non siano presenti operandi impliciti) si eseguono operazioni che non richiedono dati

Operandi

- Operandi in MC68000
  - MC68000 ha istruzioni con 0, 1 oppure 2 operandi espliciti
  - In alcune istruzioni, per alcuni operandi sono validi diversi formati di dato
  - In tali casi, la specifica del formato avviene aggiungendo al nome dell'istruzione un carattere `.` seguito da un singola lettera, detta *estensione*
  - Le estensioni possibili sono
    - `b` indica formato `byte`
    - `w` indica formato `word`
    - `l` indica formato `long`
  - Lo specificatore per un operando immediato è formato da `#` seguito dal valore dell'operando
  - Lo specificatore per una parola di registro è il nome del registro

Operandi

- Operandi in MIPS32
  - MIPS32 ha istruzioni con 0, 1, 2 oppure 3 operandi espliciti
  - In ogni istruzione, ciascun operando ha un singolo formato di dato prestabilito
  - Lo specificatore per un operando immediato è il valore dell'operando
  - Lo specificatore per una parola di registro è il nome del registro

Istruzioni di Trasferimento Dati

- In generale le istruzioni di trasferimento dati hanno 2 operandi, uno detto *sorgente*, l'altro *destinazione*
- L'effetto di queste istruzioni è copiare il contenuto dell'operando sorgente nell'operando destinazione
- In alcuni casi uno dei due operandi può essere implicito
- Sono l'equivalente dell'operazione di assegnamento di un *HLL*
- Alcune abstract machine hanno istruzioni speciali per trasferire sequenze di dati

Istruzioni di Trasferimento Dati in MC68000-ASM1

- Nelle istruzioni con 2 operandi espliciti, il primo è l'operando sorgente e il secondo è l'operando destinazione
- MC68000-ASM1 ha una istruzione di trasferimento molto generale chiamata `move` con 2 operandi espliciti
  - l'operando sorgente può essere una qualunque parola (di registro o di memoria)
  - il formato dell'operando sorgente è quello indicato dall'estensione, se presente, oppure `word` in caso contrario
  - l'operando destinazione può essere una parola di registro o memoria-ordinaria
  - se l'operando destinazione è un registro indirizzi allora ha formato `long`, altrimenti ha lo stesso formato dell'operando sorgente
- `moveq` copia di un operando immediato di formato `byte` in un registro dati

Istruzioni di Trasferimento Dati in MC68000-ASM1

- `exg` scambia il contenuto di due registri
  - entrambi gli operandi sono registri, usati con formato `long`
  - il contenuto del primo operando dopo l'esecuzione dell'istruzione è uguale al contenuto che il secondo operando aveva prima dell'esecuzione dell'istruzione, e viceversa
- `swap` scambia il contenuto delle "metà" di un registro dati
  - ha un unico operando che deve essere un registro dati, usato con formato `long`
  - per $0 \leq i \leq 15$, il bit in posizione i del registro al termine dell'esecuzione dell'istruzione contiene lo stesso valore binario che il bit in posizione $i + 16$ conteneva prima dell'esecuzione dell'istruzione
  - per $16 \leq i \leq 31$, il bit in posizione i del registro al termine dell'esecuzione dell'istruzione contiene lo stesso valore binario che il bit in posizione $i - 16$ conteneva prima dell'esecuzione dell'istruzione

Istruzioni di Trasferimento Dati in MIPS32-MARS

- In MIPS32-MARS ci sono istruzioni diverse per trasferimenti di tipo diverso
- L'istruzione `move` copia un dato tra due dei GPR
  - ha 2 operandi espliciti, il primo è l'operando destinazione, il secondo è il sorgente
  - entrambi gli operandi sono GPR, e quindi con formato `word`
- L'istruzione `li` copia un operando immediato in un GPR
  - ha 2 operandi espliciti, il primo è l'operando destinazione, il secondo è il sorgente
  - l'operando sorgente è un operando immediato, può avere formato `half` oppure `word`
  - l'operando destinazione è un GPR

Istruzioni di Trasferimento Dati in MIPS32-MARS

- L'istruzione `lui` ha due operandi, il primo è un GPR e il secondo un operando immediato in formato `half`
  - azzera i bit di posizione compresa tra 0 e 15 del primo operando
  - copia il contenuto dell'operando immediato nei bit di posizione compresa tra 16 e 31 del GPR
- Le istruzioni `mfhi`, `mflo`, `mthi`, `mtlo` hanno un solo operando esplicito, che è un GPR
  - `mfhi` copia il contenuto del registro `HI` nell'operando esplicito
  - `mflo` copia il contenuto del registro `LO` nell'operando esplicito
  - `mthi` copia il contenuto dell'operando esplicito nel registro `HI`
  - `mtlo` copia il contenuto dell'operando esplicito nel registro `LO`

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 LPS ci interessiamo prevalentemente delle prime due categorie e nella prima parte del corso consideriamo solo la prima

Istruzioni Aritmetiche in MC68000-ASM1

- La maggior parte delle istruzioni di calcolo di operazioni aritmetiche hanno 2 operandi
  - eseguono un'operazione aritmetica tra i contenuti dei due operandi e memorizzano il risultato in uno di essi, sovrascrivendo il contenuto precedente
  - tutte possono usare registri dati come operandi
  - alcune ammettono come operandi anche registri indirizzi, parole memoria-immediate o parole memoria-ordinarie
  - molte istruzioni ammettono diversi formati dato
- Vi sono alcune istruzioni a 1 operando, che eseguono un'operazione aritmetica sull'operando e memorizzano in esso il risultato

Istruzioni Aritmetiche in MC68000-ASM1

- Le istruzioni `add` e `sub` sono istruzioni a 2 operandi
  - Il primo operando
    - può essere un registro, una parola memoria-ordinaria o una parola memoria-immediata
    - il suo formato è quello indicato dall'estensione, se presente, oppure `word` in caso contrario
  - Il secondo operando
    - può essere un registro o una parola memoria-ordinaria
    - se è un registro indirizzi allora ha formato `long`, altrimenti ha lo stesso formato del primo operando
  - I due operandi non possono essere entrambi parole memoria-ordinarie
- `add` calcola la somma dei contenuti degli operandi, e la memorizza nel secondo operando
- `sub` calcola la differenza tra il contenuto del secondo operando e quello del primo, e la memorizza nel secondo operando

Istruzioni Aritmetiche in MC68000-ASM1

- L'istruzione `muls` ha due operandi
  - Il primo operando
    - può essere un registro dati (ma non indirizzi), una parola memoria-ordinaria o una parola memoria-immediata
    - il suo formato è `word`
  - Il secondo operando
    - è un registro dati
    - ha formato `long`
  - Poiché i formati degli operandi sono fissi, non ha estensione; tuttavia in versioni successive di M68000, essa ammette operandi con altri formati e quindi un'estensione
- Calcola, come valore di 32 cifre binarie, il prodotto tra
  - il numero contenuto nel primo operando
  - il numero contenuto nei bit di posizione compresa tra 0 e 15 del secondo
- Tale prodotto viene memorizzato nel secondo operando

Istruzioni Aritmetiche in MC68000-ASM1

- L'istruzione `divs` ha due operandi
  - Il primo operando ha formato `word` e può essere un registro dati (ma non indirizzi), una parola memoria-ordinaria o una parola memoria-immediata
  - Il secondo operando ha formato `long` ed è un registro dati
  - Poiché i formati degli operandi sono fissi, non ha estensione; tuttavia in versioni successive di M68000, essa ammette operandi con altri formati e quindi un'estensione
- Effettua la divisione intera tra
  - il numero di 32 cifre binarie contenuto nel secondo operando (dividendo)
  - il numero di 16 cifre binarie contenuto nel primo (divisore)
- Il quoziente viene memorizzato nei bit di posizione compresa tra 0 e 15 del secondo operando
- Il resto viene memorizzato nei bit di posizione compresa tra 16 e 31 del secondo operando

Istruzioni Aritmetiche in MC68000-ASM1

- L'istruzione `neg` ha un operando
- L'operando
  - può essere un registro dati (ma non indirizzi) o una parola memoria-ordinaria
  - il suo formato è quello indicato dall'estensione, se presente, oppure `word` in caso contrario
- L'istruzione cambia il segno al valore contenuto nell'operando

Istruzioni Aritmetiche in MIPS32-MARS

- La maggior parte delle istruzioni di calcolo di operazioni aritmetiche hanno 3 operandi
  - eseguono un'operazione tra i contenuti di due operandi e memorizzano il risultato nel terzo operando
  - tutte possono usare i GPR come operandi
  - alcune ammettono anche operandi immediati
- Le altre sono istruzioni a 2 operandi
  - gli operandi sono GPR
  - alcune istruzioni per moltiplicazione e divisione usano i registri speciali LO e HI

Istruzioni Aritmetiche in MIPS32-MARS

- Le istruzioni `add` e `sub` sono istruzioni a 3 operandi
  - I primi due operandi sono GPR
  - Il terzo può essere un GPR o un operando immediato
- `add` calcola la somma dei contenuti del secondo e terzo operando, e la memorizza nel primo operando
- `sub` calcola la differenza tra il contenuto del secondo operando e quello del terzo, e la memorizza nel primo operando

Istruzioni Aritmetiche in MIPS32-MARS

- L'istruzione `mul` ha tre operandi
  - I primi due operandi sono GPR
  - Il terzo può essere un GPR o un operando immediato
- Calcola il prodotto dei contenuti del secondo e terzo operando, come valore di 64 cifre binarie
- Le 32 cifre binarie di tale prodotto che hanno valore minore, vengono memorizzate nel primo operando
- Le altre 32 cifre binarie vengono perdute
- I registri speciali `LO` e `HI` assumono un contenuto indefinito

Istruzioni Aritmetiche in MIPS32-MARS

- L'istruzione `mult` ha due operandi, entrambi GPR
- Calcola il prodotto dei contenuti dei due operandi, come valore di 64 cifre binarie
- Le 32 cifre binarie di tale prodotto che hanno valore minore, vengono memorizzate nel registro speciale `LO`
- Le altre 32 cifre binarie vengono memorizzate nel registro speciale `HI`

Istruzioni Aritmetiche in MIPS32-MARS

- L'istruzione `div` può avere o due o tre operandi
- Istruzione `div` con tre operandi
  - I primi due operandi sono GPR
  - Il terzo può essere un GPR o un operando immediato
  - Effettua la divisione intera tra il contenuto del secondo operando (dividendo) e quello del terzo (divisore)
  - Il quoziente viene memorizzato nel primo operando
  - Il resto viene perduto
  - I registri speciali `LO` e `HI` assumono un contenuto indefinito
- Istruzione `div` con due operandi
  - I due operandi devono essere entrambi GPR
  - Effettua la divisione intera tra il contenuto del primo operando (dividendo) e quello del secondo (divisore)
  - Il quoziente viene memorizzato in `LO`
  - Il resto viene memorizzato in `HI`