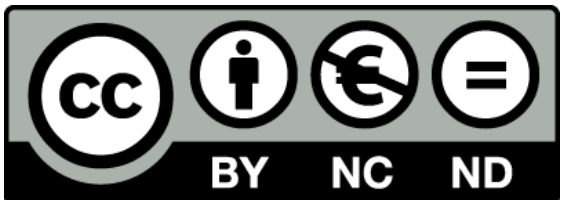
Laboratorio di Programmazione di Sistema

Fondamenti dei Linguaggi Assembly 1

Luca Forlizzi, Ph.D.

Versione 23.1



Linguaggi Assembly

- I *linguaggi macchina* (*LM*) sono i linguaggi eseguiti dalle CPU (ovvero le abstract machine di livello 2)
- Dato un programma P scritto in linguaggio C (il linguaggio di alto livello di riferimento in LPS), in LPS interessa capire la forma e la struttura di un programma P', scritto in un *LM*, che traduce correttamente P, e come P' viene eseguito da una CPU
- Come è noto, le istruzioni dei *LM* sono *stringhe binarie*, ovvero sequenze di cifre binarie
- Poiché è difficile per gli umani leggere e scrivere stringhe binarie senza commettere errori, vengono definiti dei linguaggi che hanno una semantica identica, o molto simile, a quelle dei *LM* ma i cui comandi hanno una sintassi più facilmente leggibile, scrivibile e memorizzabile dagli esseri umani, chiamati *linguaggi assembly* (*ASM*)

Linguaggi Assembly

- Esistono molteplici *LM*, diversi tra loro semanticamente
- Dato un *LM*, si definisce un corrispondente *linguaggio assembly*, le cui istruzioni hanno una semantica equivalente a quelle del LM
- Poiché sono semanticamente molto simili, studiare un linguaggio *ASM* permette di conoscere anche la semantica del corrispondente *LM*

Linguaggi Assembly

- Poiché esistono molteplici LM diversi tra loro e poiché ciascun *ASM* deve avere semantica identica o molto simile a quella del corrispondente *LM*, non è possibile definire un singolo *ASM* che corrisponda a due o più *LM* diversi tra loro
- Dunque esistono anche 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

Linguaggi Assembly

- Le architetture dei linguaggi assembly sono chiamate *Assembly Language Programming Model* (*ASM-PM*)
- 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*

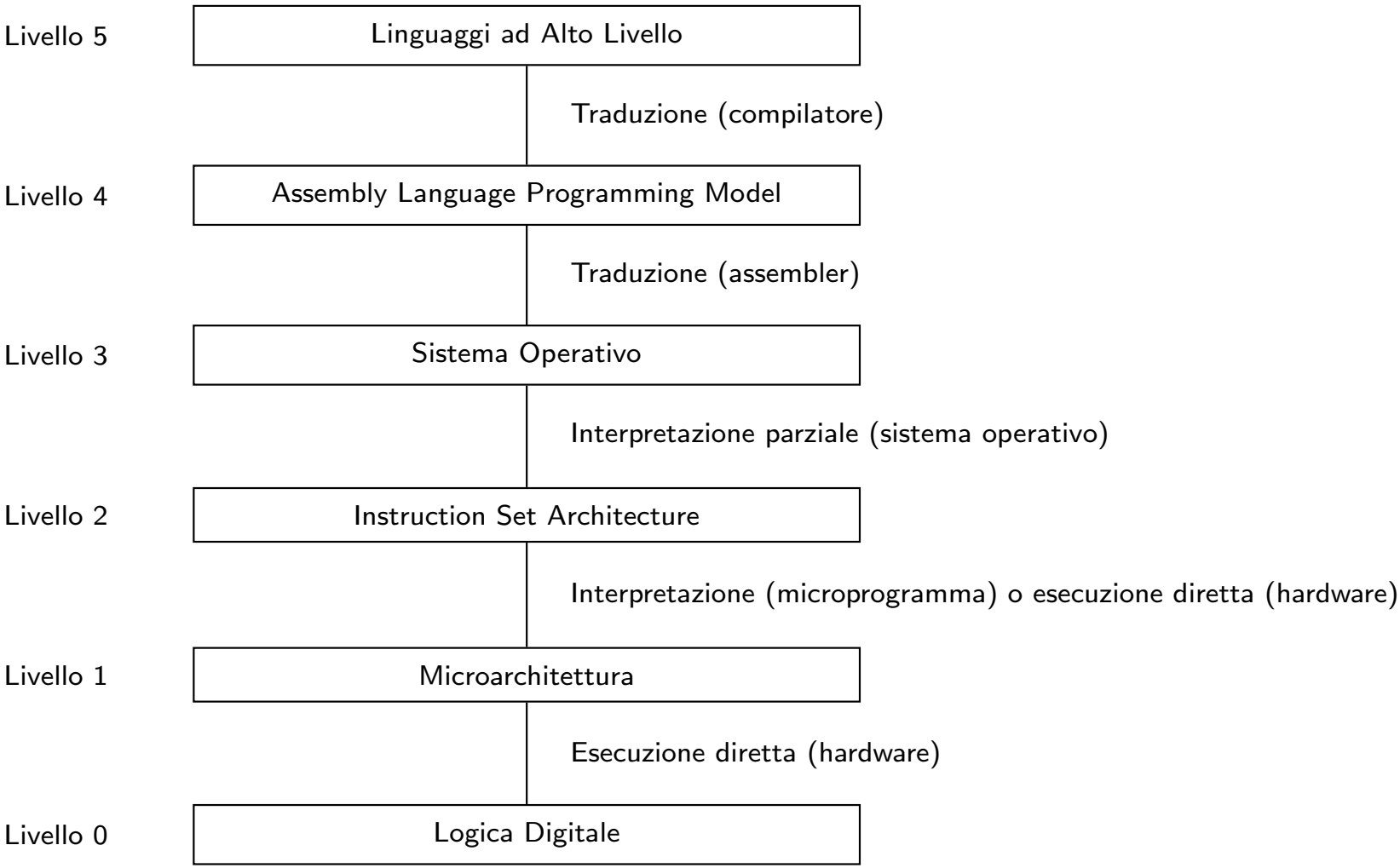
Semplificazione del modello a 6 livelli

- In LPS viene presentato un modello concettuale che definisce un computer come un dispositivo di calcolo strutturato in 6 livelli di astrazione, indicati con i numeri da 0 a 5
- Ad ogni livello troviamo una abstract machine M_i che esegue comandi di un linguaggio L_i, offrendo servizi ai livelli superiori

Semplificazione del modello a 6 livelli

- Una abstract machine di livello $i > 0$ viene implementata combinando abstract machine dei livelli sottostanti, ovvero utilizzando uno o più linguaggi L_h, dove $h < i$
- L'esecuzione di un programma scritto in un linguaggio L_i di livello i, con $i > 0$, può avvenire
  - traducendo il programma in uno equivalente scritto in un linguaggio L_h di livello $h < i$
  - oppure utilizzando un interprete scritto in un linguaggio L_h di livello $h < i$
  - se $h > 0$, per eseguire il programma scritto in L_h si ripete quanto fatto per quello scritto in L_i

Semplificazione del modello a 6 livelli



Semplificazione del modello a 6 livelli

- È importante sottolineare che gli *HLL* (linguaggi di livello 5) sono relativamente indipendenti dai linguaggi dei livelli sottostanti, mentre tra linguaggi dei livelli 2, 3 e 4 sussistono legami molto stretti
- Infatti, un programma scritto in un *HLL* può essere tradotto o interpretato in differenti linguaggi di livello inferiore
- Di solito, invece, si traduce o interpreta da un certo linguaggio di livello 4 in uno specifico linguaggio di livello 3
- Allo stesso modo, si traduce o interpreta da un certo linguaggio di livello 3 in uno specifico linguaggio di livello 2

Semplificazione del modello a 6 livelli

- Si discuterà ora come sia possibile e utile, per agevolare la comprensione dei contenuti di LPS, concentrare l'attenzione su 3 soli livelli di astrazione
- Innanzitutto, in LPS non vengono studiati i livelli 0 e 1 in quanto tipicamente eseguono programmi hardware, quindi non modificabili
- Quindi, dato un programma P scritto in linguaggio C (il linguaggio di alto livello di riferimento in LPS), interessa capire la forma e la struttura di un programma P', scritto in un LM, che traduce correttamente P, e come P' viene eseguito da un dispositivo di elaborazione di livello 2

Semplificazione del modello a 6 livelli

- In molte implementazioni del linguaggio C si ha che:
  - un programma C viene tradotto in un programma scritto in un linguaggio di livello 4 equivalente
  - il programma di livello 4 viene tradotto in un programma scritto in un linguaggio di livello 3
  - il programma di livello 3 viene parzialmente interpretato da una abstract machine di livello 2
- Poiché in LPS si concentra l'attenzione sugli aspetti fondamentali del linguaggio C, è possibile trascurare, in molti casi, anche il livello 3 (oltre ai livelli 0 e 1), per la ragione che ora verrà spiegata

Semplificazione del modello a 6 livelli

- In ciascuno dei livelli 5, 4 e 3 possiamo distinguere dei *servizi di base* e dei *servizi avanzati*
- I servizi di base sono quelli fondamentali per la scrittura di un programma, quali
  - definire strutture dati elementari
  - effettuare calcoli elementari su dati interi o floating point
  - controllare la successione delle istruzioni eseguite
- Tra i servizi comunemente considerati avanzati troviamo
  - definire strutture dati sofisticate
  - effettuare operazioni I/O
  - effettuare calcoli complessi
- Tutti i servizi offerti dal livello 2, invece, sono servizi di base: a tale livello i servizi avanzati non esistono e qualsiasi compito deve essere svolto con complessi programmi che usano solo servizi basilari

Semplificazione del modello a 6 livelli

- A grandi linee troviamo una corrispondenza concettuale tra le componenti dei linguaggi ai diversi livelli

Livello	Servizi base	Servizi avanzati
5	C Language	C Library
4	*ASM*	*ASM-API*
3	*LM**	*system call*
2	*LM*	

Semplificazione del modello a 6 livelli

- La corrispondenza concettuale tra i linguaggi si rispecchia in come vengono effettuate la traduzione o l'interpretazione

Livelli	Costrutto	Traduzione/interpretazione
$5 \rightarrow 4$	costrutto C Language	programma *ASM*, di rado qualche comando *ASM-API*
$5 \rightarrow 4$	costrutto C Library	programma *ASM* e *ASM-API*
$4 \rightarrow 3$	singola istruzione *ASM*	poche (tipicamente da 1 a 3) istruzioni *LM**
$4 \rightarrow 3$	comando *ASM-API*	programma *LM** e *system call*
$3 \rightarrow 2$	singola istruzione *LM**	una istruzione *LM*
$3 \rightarrow 2$	singola *system call*	programma *LM*

Semplificazione del modello a 6 livelli

- Si ha una corrispondenza analoga tra le componenti delle architetture ai diversi livelli

Livello	Servizi base	Servizi avanzati
5	freestanding C Standard	parti di hosted C Standard non contenute in freestanding C Standard
4	*ASM-PM*	*ASM-API architecture*
3	*ISA**	*operating system architecture*
2	*ISA*	

- In LPS ci si concentra sulle componenti dei linguaggi che offrono i servizi base, mentre quelle che offrono servizi avanzati verranno studiate in *Sistemi Operativi con Laboratorio*
- Proprio il fatto che in LPS si studiano in prevalenza servizi base, permette, di solito, di non considerare il livello 3
- La ragione dipende dalla particolare relazione tra i livelli 3 e 2

Semplificazione del modello a 6 livelli

- Il livello 3, del sistema operativo, è un livello ibrido, in quanto i servizi di base che esso offre sono simili (in molti casi uguali) a quelli del livello 2
- Relazione tra LM^* e LM
  - nella maggioranza dei casi, un'istruzione di LM^* è identica, sia nella sintassi che nella semantica, ad un'istruzione LM
  - vi sono rare eccezioni
    - alcune istruzioni di LM^* sono istruzioni di LM con leggere modifiche determinate dal sistema operativo
    - alcune istruzioni di LM non sono disponibili in LM^*
- Trascurando le eccezioni, si può dire che il livello 3 "espone" il livello 2, rendendolo accessibile dai livelli superiori

Semplificazione del modello a 6 livelli

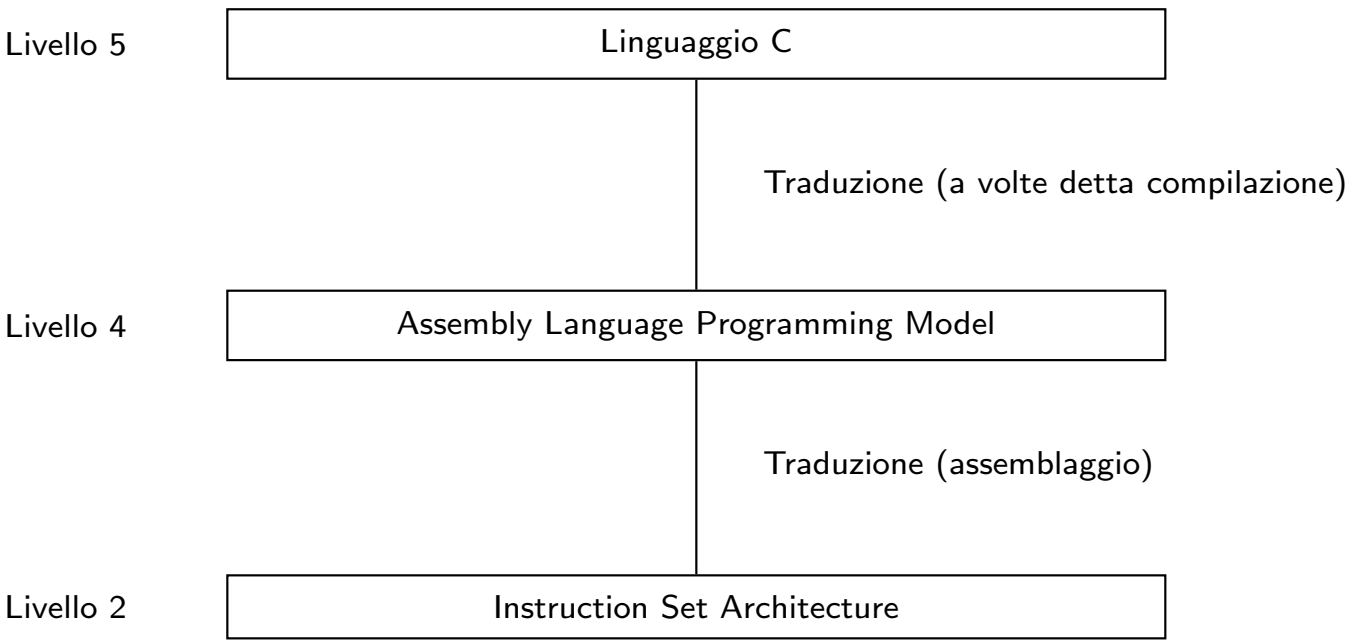
- Dunque, in relazione al livello 3, si ha la seguente situazione
  - in LPS non vengono studiati i servizi avanzati del livello 3, quindi non serve studiare la componente *system call* di un linguaggio di livello 3
  - un LM^* (ovvero la componente che fornisce i servizi base di un linguaggio di livello 3) è uguale o molto simile a un LM, quindi è sufficiente studiare LM
- Questa situazione giustifica il fatto che in LPS si trascura, nella maggior parte delle situazioni, il livello 3
- Ciò equivale a considerare computer privi di sistema operativo, cosa che spesso si verifica anche in pratica, nel mondo dei sistemi embedded

Modello a 3 Livelli

- È stato introdotto un modello concettuale di computer come dispositivo di calcolo strutturato in 6 livelli di astrazione
- Tuttavia in LPS non vengono studiati i livelli della Logica Digitale, della Microarchitettura e del Sistema Operativo
- In questa situazione è utile prendere in considerazione anche un modello di computer semplificato composto da soli 3 livelli di astrazione

Modello a 3 Livelli

- Quando non diversamente specificato, quindi, si farà riferimento ad un modello di computer semplificato composto da soli 3 livelli di astrazione, che usa il Linguaggio C come linguaggio di livello 5



Modello a 3 Livelli

- Come abbiamo detto, il linguaggio C e la sua architettura sono indipendenti dagli *ASM* e dagli *ASM-PM*
  - Un programma C può essere tradotto o interpretato mediante *ASM* diversi
  - Anzi, in linea di principio un programma C potrebbe essere tradotto o interpretato mediante qualunque *ASM*
- Al contrario, ogni *ASM-PM A corrisponde* 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* corrispondente a tale *ISA*
- Poiché un *ASM-PM* e la corrispondente *ISA* 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

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:

RUN	*M* esegue continuamente istruzioni, in modo sequenziale o parallelo
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
OFF	*M* è spenta

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, fino a 2^{24} byte di memoria principale
1982	MC68010	• supporto per la virtualizzazione potenziato
1984	MC68020	• fino a 2^{32} byte di memoria principale • 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 la *ISA*, è del tutto indipendente dalle altre caratteristiche, in quanto esse non hanno impatto sul *LM* della CPU; altre implementazioni di MC68000 possono differire da `MC68000P8` relativamente a tali 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" [PH]
- Il Prof. Hennessy, uno degli autori di [PH], è stato tra i fondatori di MIPS Computer Systems Inc.

MIPS

- Principali versioni di MIPS

Anno	Nome	Lunghezza di 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`