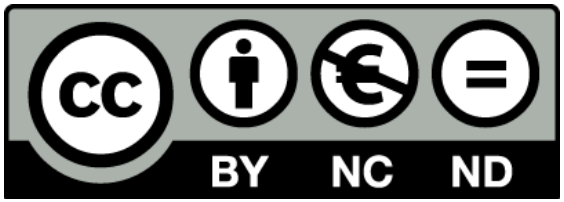
Laboratorio di Programmazione di Sistema

Fondamenti dei Linguaggi Assembly 3

Luca Forlizzi, Ph.D.

Versione 23.1



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*

- Tipicamente, 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*

- La maggior parte dei linguaggi *ASM* ha una struttura sintattica semplice, che descriviamo nel seguito
- I costrutti presenti nei linguaggi *ASM* si possono classificare in base alle seguenti categorie

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

  Direttive : comandi eseguiti durante la traduzione

  Istruzioni : comandi eseguiti durante l'esecuzione

  Commenti : testo privo di significato per la *abstract machine* e quindi da essa ignorato, che 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*

- 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
- Quindi ogni riga può contenere, nell'ordine indicato, i seguenti 3 costrutti, tutti opzionali:
  1. una definizione di label
  2. una direttiva oppure un'istruzione
  3. un commento

Direttive e Istruzioni

- Direttive e istruzioni sono i comandi dei linguaggi *ASM*
  - Le direttive vengono eseguite durate la traduzione del programma, sono quindi comandi statici
  - Le istruzioni vengono eseguite durante l'esecuzione del programma, sono quindi comandi dinamici
- La sintassi generale di direttive e istruzioni è la seguente **Nome Specificatore**$_1$,**Specificatore**$_2$, . . . ,**Specificatore**$_N$
- **Nome** è una stringa detta *nome* dell'istruzione o direttiva
- Per ogni i, **Specificatore**$_i$ è una stringa detta *specificatore di parametro* nel caso delle direttive e *specificatore di operando* nel caso delle istruzioni

Direttive e Istruzioni

- Lo stile dei nomi delle direttive e delle istruzioni tende ad essere criptico: si usano di solito nomi di massimo 4 caratteri alfabetici, 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)
- Alcuni nomi di istruzioni o direttive contengono anche caratteri numerici o speciali

Direttive e Istruzioni

- Gli specificatori di parametro e gli specificatori di operando sono stringhe formate da caratteri alfanumerici o speciali
- Gli specificatori di parametro e in quelli di operando possono contenere, in certi casi, valori numerici
- I valori numerici, in molti *ASM*, possono essere specificati con diverse rappresentazioni numeriche
- Le rappresentazioni numeriche più comuni sono
  - decimale: comoda in molte applicazioni
  - ottale: permette di rappresentare in forma compatta una stringa binaria, in quanto vi è una corrispondenza biunivoca tra sequenze di 3 cifre binarie e singole cifre ottali
  - esadecimale: permette di rappresentare in forma compatta una stringa binaria, in quanto vi è una corrispondenza biunivoca tra sequenze di 4 cifre binarie e singole cifre esadecimali

Traduzione di un Programma *ASM*

- Un assembler traduce un programmi *ASM* leggendo il codice sorgente in ordine testuale
  - 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 la traduzione del codice sorgente
  - 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 in MC68000

- 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 in MIPS32

- Alcune direttive di MIPS32-MARS 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 indica alla abstract machine di eseguire, su alcuni dati, una determinata operazione che produce alcuni risultati
- I dati su cui la abstract machine esegue l'operazione specificata dall'istruzione, sono i contenuti di alcune parole
- I risultati, dopo essere stati calcolati dall'operazione, vengono memorizzati in alcune parole

Istruzioni

- Le parole che contengono i dati e le parole in cui vengono memorizzati i risultati, sono chiamate *operandi* dell'istruzione
  - Gli operandi in cui vengono memorizzati i risultati sono chiamati *operandi destinazione*; ovviamente memorizzando una stringa binaria in un operando destinazione, si sovrascrive il precedente contenuto di tale operando
  - Gli operandi il cui contenuto non viene modificato dall'istruzione, sono chiamati *operandi sorgente*

Istruzioni

- Per ora, si può schematizzare l'esecuzione di un'istruzione come un processo articolato nelle seguenti fasi
  1. *lettura* dei dati contenuti negli operandi
  2. *esecuzione* dell'operazione
  3. *scrittura* dei risultati negli operandi destinazione
- Nel proseguo di LPS si approfondirà il processo di esecuzione di un'istruzione, mostrando che vi sono altre fasi importanti, come si è già studiato in *Architettura degli Elaboratori*

Istruzioni

- Più in dettaglio, un'istruzione specifica le seguenti informazioni
  - l'operazione da eseguire
  - la quantità di operandi
  - per ciascun operando
    - il formato di dato dell'operando
    - l'interpretazione di dato dell'operando
    - *modo di indirizzamento* dell'operando, che specifica con quale regola e con quali informazioni determinare l'identificativo dell'operando
- Molte di queste informazioni sono specificate da una stringa alfanumerica detta *nome* dell'istruzione
- Le rimanenti, relative ad alcuni degli operandi, da espressioni sintattiche chiamate *specificatori di operando*

Istruzioni

- In particolare, il nome di un'istruzione specifica
  - l'operazione da eseguire
  - la quantità di operandi
  - interpretazione di dato di ciascuno degli operandi
  - formato di dato di alcuni degli operandi
  - modo di indirizzamento di alcuni degli operandi
- Gli operandi per i quali tutte le informazioni sono specificate dal nome dell'istruzione, vengono detti operandi *impliciti*, proprio perché tutto ciò che li riguarda è implicito nel nome dell'istruzione
- Gli operandi impliciti non vengono neppure mostrati dalla sintassi dell'istruzione, in quanto non vi è necessità di farlo

Istruzioni

- Gli operandi che non sono impliciti, vengono detti *espliciti*
- Per ogni operando esplicito, l'istruzione ha uno specificatore di operando che indica il modo di indirizzamento di tale operando
- In alcuni casi poco comuni, lo specificatore di operando indica, oltre al modo di indirizzamento, anche il formato dell'operando

Operazioni Eseguite dalle Istruzioni

- Il nome di ogni istruzione indica l'operazione eseguita dall'istruzione
- Salvo rari casi, le operazioni eseguite da una istruzione *ASM* sono piuttosto semplici
- Le istruzioni vengono suddivise in tipologie, proprio in base alle operazioni eseguite
- Le tipologie di istruzione più comuni e importanti sono:
  - istruzioni di trasferimento dati
  - istruzioni aritmetiche
  - istruzioni logiche e di manipolazione di bit
  - istruzioni per il controllo della sequenza di istruzioni da eseguire
  - istruzioni di gestione della CPU e del sistema
- In questa presentazione, si introdurranno le prime due tipologie di istruzione, di altre si parlerà in seguito

Quantità di Operandi delle Istruzioni

- Ogni istruzione, indicata dal nome, ha regole precise sulla quantità di operandi ammissibili
  - Quasi tutte le istruzioni *ASM* hanno un numero di operandi prestabilito e fisso
  - Vi sono alcune eccezioni
- Tipicamente, ogni istruzione ha un numero di operandi compreso tra 0 e 3

Quantità di Operandi delle Istruzioni

- Le istruzioni a 3 operandi permettono di eseguire un'operazione tra i contenuti di due parole, memorizzando il risultato in una terza parola, in modo analogo a quanto avviene nell'espressione unstructured C
  `V = A op1 B`
  dove `V` è una variabile, `A` e `B` sono ciascuno una variabile oppure una costante e `op1` è un operatore

Quantità di Operandi delle Istruzioni

- Con 2 operandi, di solito vengono realizzate
  - operazioni tra i contenuti di due parole, memorizzando il risultato in una di tali parole, analogamente all'espressione unstructured C
    `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 sul contenuto di una parola, memorizzando il risultato in una diversa parola, analogamente all'espressione unstructured C
    `V = op3 A`
    dove `V` è una variabile, `A` è una variabile oppure una costante e op3 un operatore unario

Quantità di Operandi delle Istruzioni

- Con 1 operando si possono realizzare operazioni che modificano il contenuto di una parola, analogamente all'espressione unstructured C
  `V = op4 V`
  dove `V` è una variabile e `op4` un operatore unario
- Con 0 operandi si eseguono operazioni che non richiedono dati

Formati e Interpretazioni di Dato per gli Operandi

- Interpretazioni di dato per gli operandi
  - Ogni istruzione, stabilisce l'interpretazione di dato di ciascuno dei suoi operandi
  - Per ciascun operando vi è, di solito, una sola interpretazione di dato possibile
  - Le interpretazioni di dato, essendo determinate dal nome dell'istruzione, non sono espresse dalla sintassi
- Formati di dato per gli operandi
  - Ogni istruzione, stabilisce il formato di dato di ciascuno dei suoi operandi
  - In alcuni *ASM-PM*, per ciascun operando vi è, di solito, un solo formato di dato possibile, che quindi non viene espresso dalla sintassi
  - In altri, per alcuni operandi vi sono due o più diversi formati di dato che è possibile specificare, attraverso la sintassi dell'istruzione

Indirizzamento

- Definiamo *indirizzamento* di una parola, l'operazione che consiste in
  1. determinare l'identificativo della parola
  2. individuare la parola, usando l'identificativo e il formato
  3. accedere alla parola, in lettura o in scrittura
- La parola *indirizzamento* richiama il fatto che per le parole di memoria, che sono la maggior parte, gli identificativi sono indirizzi di memoria
- Diciamo che *indirizzare* una parola vuol dire effettuare l'indirizzamento di tale parola
- Durante l'esecuzione di una istruzione, vengono indirizzati gli operandi dell'istruzione

Modo di Indirizzamento

- Si definisce *modo di indirizzamento* (*addressing mode*) una regola che, a partire da alcune informazioni, permette di indirizzare una parola
- Ogni *ASM-PM*, definisce una certa varietà di modi di indirizzamento
- Le regole semantiche di ciascuna istruzione indicano, per ciascuno degli operandi, quali sono i modi di indirizzamento che è possibile utilizzare per tale operando

Modo di Indirizzamento

- Nella prima parte di LPS saranno sufficienti 3 modi di indirizzamento
  - *Indirizzamento implicito*
  - *Indirizzamento immediato*
  - *Indirizzamento diretto-registro*
- Altri modi di indirizzamento verranno introdotti in seguito

Indirizzamento Implicito

- Gli operandi impliciti vengono indirizzati con un modo di indirizzamento implicito
- Un operando implicito è quindi una determinata parola di memoria o di registro, identificata dalla semantica dell'istruzione
- Il programmatore, quindi, non ha la possibilità di scegliere quale parola sarà l'operando implicito
- Per questo, gli operandi impliciti non vengono, di solito, nominati esplicitamente dalla sintassi

Indirizzamento Immediato

- I modi di indirizzamento immediati permettono di indirizzare, in modo particolarmente efficiente, una parola di memoria che contiene un dato specificato dal programmatore
- Se I è un'istruzione che usa un indirizzamento immediato, ad I viene associata una specifica parola di memoria P
  - l'identificativo di P è determinato in modo automatico dall'*ASM-PM*, quindi
    - non può essere scelto dal programmatore (ma ciò non è una limitazione)
    - viene calcolato automaticamente e in modo molto efficiente
  - nessuna istruzione diversa da I può accedere a P con un modo di indirizzamento immediato
  - l'esecuzione di I non può modificare il contenuto di P, ovvero I accede a P in sola lettura

Indirizzamento Immediato

- Un operando indirizzato con l'indirizzamento immediato viene detto *operando immediato*
- Un operando immediato di un'istruzione non è mai un operando destinazione, in quanto non può essere modificato dall'istruzione
- Poiché, come abbiamo detto, usando l'indirizzamento immediato il programmatore può specificare il contenuto dell'operando ma non l'identificativo della parola che lo memorizza, lo specificatore di operando contiene una rappresentazione della stringa binaria che costituisce il contenuto dell'operando
- Tale rappresentazione può essere il valore numerico che si ha adottando una determinata interpretazione di dato

Indirizzamento Diretto-Registro

- I modi di indirizzamento diretto-registro permettono di indirizzare una parola di registro
- Se un operando viene indirizzato con l'indirizzamento diretto-registro, viene detto *operando registro*
- Lo specificatore dell'operando indica l'identificativo di un registro, ovvero il nome del registro

Operandi in MC68000

- MC68000 ha istruzioni con 0, 1 oppure 2 operandi espliciti
- Per alcune istruzioni, si può specificare il formato di dato, tra alcuni possibili, di alcuni degli operandi
- In tali casi, la specifica del formato avviene modificando il nome dell'istruzione con l'aggiunta di 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`

Operandi in MC68000

- Per ciascun operando esplicito, uno specificatore di operando indica il modo di indirizzamento e tutte le informazioni per determinare l'identificativo dell'operando
  - Lo specificatore per un operando immediato è formato da # seguito dal contenuto dell'operando
  - Lo specificatore per un operando registro è il nome del registro

Esempio MC68000: operandi con l'istruzione add

```
* istruzione 1: formato word per entrambi gli operandi
*  primo operando: indirizzamento immediato
*  secondo operando: indirizzamento diretto-registro
 add.w  #16,d2
* istruzione 2: formato long per entrambi gli operandi
*  indirizzamento diretto-registro per entrambi gli operandi
 add.l  a0,d3
```

Operandi in MIPS32

- MIPS32 ha istruzioni con 0, 1, 2 oppure 3 operandi espliciti
- Per alcune istruzioni, si può specificare se il formato di dato degli operandi immediati sia `half` oppure `word`
- In tali casi, la specifica del formato avviene in base al valore numerico v indicato nello specificatore dell'operando immediato
  - se v può essere rappresentato mediante una stringa di 16 cifre binarie, il formato dell'operando è `half`
  - altrimenti, il formato dell'operando è `word`
- In tutti gli altri casi, per ogni operando vi è un solo formato di dato possibile

Operandi in MIPS32

- Per ciascun operando esplicito, il modo di indirizzamento e tutte le informazioni per determinare l'identificativo dell'operando sono indicati da uno specificatore di operando
  - Lo specificatore per un operando immediato è il contenuto dell'operando
  - Lo specificatore per un operando registro è formato dal carattere $ seguito dal nome di un registro

Esempio MIPS32: operandi con l'istruzione add

```
* formato word per i primi due operandi
* formato half per il terzo operando, in quanto 16 può
*   essere rappresentato nelle half
* primo operando: indirizzamento diretto-registro
* secondo operando: indirizzamento diretto-registro
* terzo operando: indirizzamento immediato
 add    $t0,$s2,16
```

Istruzioni di Trasferimento Dati

- Tipicamente, le istruzioni di trasferimento dati hanno un operando *sorgente* e un operando *destinazione*
- L'effetto di queste istruzioni è copiare il contenuto dell'operando sorgente nell'operando destinazione
- L'operando sorgente non viene modificato
- 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, con una singola istruzione, i contenuti di molteplici parole

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
  - per l'operando sorgente si possono utilizzare quasi tutti i modi di indirizzamento, compresi l'indirizzamento registro-diretto e quello immediato
  - il formato dell'operando sorgente è quello indicato dall'estensione, se presente, oppure `word` in caso contrario
  - per l'operando destinazione si possono utilizzare molti modi di indirizzamento, compreso l'indirizzamento registro-diretto, ma non si può utilizzare l'indirizzamento immediato
  - se l'operando destinazione è un registro indirizzi allora ha formato `long`, altrimenti ha lo stesso formato dell'operando sorgente

Istruzioni di Trasferimento Dati in MC68000-ASM1

- `clr` azzera il contenuto dell'unico operando dell'istruzione
  - per l'operando sono ammessi molti modi di indirizzamento, tra cui il modo registro-diretto su un registro dati, ma non il modo immediato né quello registro-diretto su un registro indirizzi
  - il formato dell'operando è quello indicato dall'estensione, se presente, oppure `word` in caso contrario
- `exg` scambia il contenuto di due registri
  - entrambi gli operandi sono registri, indirizzati con il modo registro-diretto, e entrambi sono 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

Istruzioni di Trasferimento Dati in MC68000-ASM1

- `swap` scambia il contenuto delle "metà" di un registro dati
  - ha un unico operando che deve essere un registro dati, indirizzato con il modo registro-diretto e usato con formato `long`
  - al termine dell'esecuzione dell'istruzione, per $0 \leq i \leq 15$, il bit in posizione i del registro contiene la cifra binaria che il bit in posizione $i + 16$ conteneva prima dell'esecuzione dell'istruzione
  - al termine dell'esecuzione dell'istruzione, per $16 \leq i \leq 31$, il bit in posizione i del registro contiene la cifra binaria 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 sono indirizzati con il modo registro-diretto e hanno 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 indirizzato con modo immediato e 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 operando implicito, che è `HI` o `LO`, e 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

- Le istruzioni aritmetiche interpretano i dati come valori numerici
- Si possono classificare in
  - Istruzioni di calcolo di un'operazione aritmetica tra valori interi
  - Istruzioni di confronto tra valori interi
  - Istruzioni di calcolo di un'operazione tra valori floating point
  - Istruzioni di confronto tra valori floating point
- In LPS studiamo solo le istruzioni delle prime due categorie, che interpretano i contenuti dei loro operandi come numeri interi
- In questa presentazione descriviamo solo istruzioni della prima categoria

Istruzioni Aritmetiche in MC68000-ASM1

- La maggior parte delle istruzioni di calcolo di operazioni aritmetiche hanno 2 operandi
  - eseguono un'operazione aritmetica tra i valori dei due operandi e memorizzano il risultato in uno di essi, sovrascrivendo il contenuto precedente
  - tutte possono usare come operandi parole contenute in registri dati
  - alcune ammettono come operandi anche parole di memoria o contenute in registri indirizzi
  - 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
    - ammette quasi tutti i modi di indirizzamento, compresi i modi registro-diretto e immediato
    - ha il formato indicato dall'estensione, se presente, oppure `word` in caso contrario
  - Il secondo operando
    - ammette quasi tutti i modi di indirizzamento, compreso il modo registro-diretto, ma non il modo immediato
    - se è una parola di un registro indirizzi allora ha formato `long`, altrimenti ha lo stesso formato del primo operando
  - Se entrambi gli operandi sono parole di memoria, il primo deve necessariamente usare l'indirizzamento immediato
- `add` calcola la somma dei valori degli operandi, e la memorizza nel secondo operando
- `sub` calcola la differenza tra il valore 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
    - ammette quasi tutti i modi di indirizzamento, compresi i modi immediato e registro-diretto, ma in quest'ultimo caso il registro indicato deve essere un registro dati (non indirizzi)
    - ha formato `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 valore del primo operando
  - il valore 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 ammette quasi tutti i modi di indirizzamento, compresi i modi immediato e registro-diretto, ma in quest'ultimo caso il registro indicato deve essere un registro dati (non indirizzi)
  - 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
  - ammette diversi modi di indirizzamento, tra cui il modo registro-diretto su un registro dati, ma non il modo immediato né quello registro-diretto su un registro indirizzi
  - ha il formato indicato dall'estensione, se presente, oppure `word` in caso contrario
- L'istruzione cambia il segno al valore dell'operando

Istruzioni Aritmetiche in MIPS32-MARS

- La maggior parte delle istruzioni di calcolo di operazioni aritmetiche hanno 3 operandi
  - eseguono un'operazione tra i valori di due operandi e memorizzano il risultato nell'altro 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 specific purpose 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, di formato `half` oppure `word`
- `add` calcola la somma dei valori del secondo e terzo operando, e la memorizza nel primo operando
- `sub` calcola la differenza tra il valore 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, di formato `half` oppure `word`
- Calcola il prodotto dei valori del secondo e terzo operando, come valore rappresentato da una stringa di 64 cifre binarie
- Le 32 cifre binarie di tale prodotto che hanno posizioni minori, vengono memorizzate nel primo operando
- Le altre 32 cifre binarie vengono perdute
- I registri `LO` e `HI` assumono un contenuto indefinito

Istruzioni Aritmetiche in MIPS32-MARS

- L'istruzione `mult` ha due operandi, entrambi GPR
- Calcola il prodotto dei valori dei due operandi, come valore rappresentato da una stringa di 64 cifre binarie
- Le 32 cifre binarie di tale prodotto che hanno posizioni minori, vengono memorizzate nel registro `LO`
- Le altre 32 cifre binarie vengono memorizzate nel registro `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, di formato `half` oppure `word`
  - Effettua la divisione intera tra il valore del secondo operando (dividendo) e quello del terzo (divisore)
  - Il quoziente viene memorizzato nel primo operando
  - Il resto viene perduto
  - I registri `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 valore del primo operando (dividendo) e quello del secondo (divisore)
  - Il quoziente viene memorizzato in `LO`
  - Il resto viene memorizzato in `HI`

Istruzioni Aritmetiche in MIPS32-MARS

- L'istruzione `rem` ha tre operandi
  - I primi due operandi sono GPR
  - Il terzo può essere un GPR o un operando immediato, di formato `half` oppure `word`
- Calcola il resto della divisione intera tra il valore del primo operando (dividendo) e quello del secondo (divisore)
- Tale risultato viene memorizzate nel primo operando
- I registri `LO` e `HI` assumono un contenuto indefinito